НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 122 Комп’ютерні науки та інформаційні технології

на тему: Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

Виконав: студент 4-го курсу, групи ТР-62

_____Гученко Микита Сергійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ст. викл., Гайдаржи В.І._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Гученку Микиті Сегрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

керівник роботи Гайдаржи Володимир Іванович, ст.викл.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 2020 р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати задачу створення web-системи з централізованого управління розповсюдженням та опрацюванням навчальної літератури.

5. Перелік ілюстративного матеріалу

схеми архітектури додатку, знімки екранних форм, діаграма прецедентів системи, діаграма структури системи, зразки розробленого інтерфейсу додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”11” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка структур окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного продукту		
8.	Передзахист		
9.	Захист		

Студент _____ _____Гученко М. С._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Гайджаржи В. І._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дана дипломна робота присвячена розробці web-системи з централізованого управління розповсюдженням та опрацюванням навчальної літератури.

Метою роботи є централізоване управління розповсюдженням та опрацюванням навчальної літератури за допомогою web-системи.

Для досягнення мети були вирішені наступні задачі:

1. Проведено аналіз необхідної функціональності системи.
2. Спроектовано чітку та узгоджену архітектуру web-системи.
3. Розроблено API з дотриманням REST принципів.
4. Розроблено та протестовано інформаційну систему.
5. Проведено інтеграцію з хмарним сховищем.
6. Web-систему розгорнуто в хмарній платформі Heroku.

Робота має 3 апробації.

Ключові слова: поширення ресурсів, RESTful API, навчальна література, навчальний процес, хмарне сховище.

Обсяг звіту становить 70 сторінок, міститься 16 ілюстрацій. Загалом опрацьовано 17 джерел.

ABSTRACT

This thesis is devoted to the development of a web-system for centralized management of distribution and processing of educational literature.

The purpose of the work is the centralized management of distribution and processing of educational literature using a web-system.

To achieve the goal, the following tasks were solved:

1. The analysis of the necessary functionality of the system was performed.
2. Designed clear and consistent architecture of the web-system.
3. Developed API with REST principles.
4. The information system was developed and tested.
5. Integration with cloud storage was built.
6. The web-system is deployed on the Heroku cloud platform.

The work has 3 approbations.

Keywords: resource distribution, RESTful API, educational literature, learning process, cloud storage.

Scope of the report is 70 pages contain 16 illustrations. Generally18 sources have been processed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI	–	User interface
REST	–	Representational State Transfer,
API	–	Application Program Interface
JSON	–	JavaScript Object Notation
ЕК КПІ	–	Електронний кампус НТУУ “КПІ”

ВСТУП

Об’єми інформації та швидкість з якою, ці дані необхідно оновлювати для підтримки актуальності невпинно зростає. Технологічні проблеми навчального процесу вимагають наявності інформаційної системи для зберігання та розповсюдження навчальної літератури, що має сприяти покращенню навчального процесу.

Існуючі інструменти у своїй більшості пропонують рішення [1], що вимагають багато часу та ресурсів на впровадження, що не кожен навчальний заклад може собі дозволити. Огляд таких рішень пропонується у першому розділі. Однією з поставлених цілей було створити систему, що автоматизує процес поширення навчальної літератури, зробить його простим у використані та легким у впроваджені. Окремо слід підкреслити, що для рішення поставлених цілей було розглянуто окрім конкретних інформаційних систем, цілі категорії систем, адже практика показує, що деякі навчальні заклади використовують для вирішення деяких з проблем, поставлених у дипломній роботі, онлайн платформи для публікування навчальних курсів або просто використовують хмарні сховища. У деяких випадках використання хмарних сховищ відбувається неявним чином, а саме через електроні листи. У наступному розділі обумовлено чому ці підходи не вирішують поставлених проблем.

Перш за все розроблювана система знімає з викладача відповідальність в управлінні розсилкою навчальних матеріалів. Система бере на себе задачу в потрібний день відправляти студентам книги, конспекти, лабораторні роботи та інші навчальні матеріали. Система підвищує якість та швидкість опанування студентами навчальних матеріалів, у порівнянні з традиційними підходами.

В цій дипломній роботі викладено повний перелік методів та підходів використаних в ході проектування, розробки, тестування, та розгортання програмного застосунку. Вибір архітектури інформаційної системи є одним з найважливіших питань. Це сприяло до того щоб виокремити розділ під аналіз архітектури застосунку де ґрунтовно обумовлено чому саме вибраний підхід є найкращим.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Метою даної дипломної роботи є централізоване управління розповсюдженням та опрацюванням навчальної літератури за допомогою web-системи.

1.1 Огляд предметної області

Технологічні проблеми навчального процесу вимагають наявності інформаційної системи для зберігання та розповсюдження навчальної літератури, що має сприяти покращенню навчального процесу.

Зберігання інформаційних ресурсів фундаментальна проблема вітчизняного і світового документознавства. В умовах інформатизації суспільства вона потребує якісно нового наукового осмислення. Інформаційні ресурси, що використовуються навчальними закладах потребують додатковий інструментарій, а точніше необхідним є створення системи, здатної забезпечити інтеграцію навчальних матеріалів. Для підвищення ефективності їх використання необхідно налагодити роботу з метаданими цих ресурсів.

1.2 Поставлені проблеми

Основний фокус стоїть на покращення користування навчальними матеріалами студентами з мінімальною ціною впровадження системи. Основний приціл стоїть на студентів, що навчаються на стаціонарній основі, та потребують регулярної взаємодії з викладачем.

При розробці відповідної інформаційної системи потрібно розв’язати наступні проблеми:

- відсутність можливості заздалегідь розпланувати дату поширення навчальних матеріалів,

- відсутність автоматизованих засобів для викладача формування масиву навчальної літератури (наповнення електронних листів ідентифікаторами, посилань на тему з навчального плану),
- відсутність контролю процесу опанування навчальної літератури,
- визначення спрямованості навчальних матеріалів отриманих студентом з плином часу,
- відсутність організаційних принципів збереження навчальних матеріалів.

Важливо розглянути ряд проблем більш детально.

1.2.1 Вартість впровадження

Існуючі рішення вимагають досить витрат по часу і грошей для впровадження. Це може дозволити не кожен навчальний заклад, тому вартість впровадження є важливою характеристикою для розроблюваної системи.

Розроблена інформаційна система дозволяє його легко впровадити за рахунок того, що студенту необов'язково користуватися системою, достатньо мати пошту, на яку буде приходити вся необхідна інформація з контентом частково згенерував системою і частково викладачем.

1.2.2 Поширення навчальних матеріалів

У більшій мірі коли мова йде про комунікацію між вчителем та студентами у перспективі поширення навчальних матеріалів, цей процес проходить через електрону пошту. Зазвичай, для таких цілей створюється одна поштова адреса на групу, на яку в майбутньому викладачі будуть відправляти матеріали та додаткову інформацію.

Виникає низка проблем:

1. Викладач зобов’язаний мати електроні адреси груп яким він хоче відправляти навчальні матеріали та іншу інформацію.
2. Обмеження на розмір електронних ресурсів. Послуги електронної пошти обмежують розмір файлів, що прикріплених до листа. Наприклад Gmail має верхню межу для розміру файлу у 25 Mb, станом на час написання диплому. Якщо прикріплений файл більше 25 Мб, він завантажується на Google Drive,

а Gmail розміщує посилання для завантаження на файл у тілі повідомлення електронного листа. Враховуючи те, що зазвичай електрона пошта викладача є особистою немає гарантії того, що матеріал завантажений на Google Drive не буде видалений, чи обмежений у доступі під час всього процесу навчання студента.

3. Досить одному студенту випадково або невипадково видалити електронний лист від викладача і вся група не матиме навчальних матеріалів чи важливої інформації.
4. На викладачеві лежить відповідальність за те щоб кожна електрона адреса була додана в розсилку матеріалів.
5. Викладач не розуміє хто саме написав йому листа, якщо студентом використовується спільна електрона адреса.
6. Відсутність заздалегідь розписати план відправлення навчальних матеріалів.
7. По електронній поштовій адресі, як правило важко сказати хто є відправником, адже не все використовують підписи.

У розроблюваній системі описані проблеми вирішуються за допомогою наступних можливостей:

1. Викладач має можливість один раз розписати план поширення навчальних матеріалів для цілого семестру.
2. Замість загальної електронної пошти використовується електрона пошта кожного студента. Для цього викладачу необхідно лише вибрати необхідну групу і система включить кожного студента у розсилку.
3. Контакти студентів та викладачів структуровані та доступні у зручній формі. Студентам стає набагато простіше знайти потрібний канал зв’язку з викладачем у разі необхідності.
4. Система автоматично додає до листа інформацію про відправника.

1.2.3 Відсутність єдиного підходу до оформлення електронних листів

З плином часу на електрону пошту потрапляє велика кількість листів. Просте коли приходить час переглянути матеріали, що буди відправлені декілька років, це стає дуже проблематично. Причина цьому – відсутність систематизації.

Розроблювана інформаційна система використовує єдину структуру для електронних листів. Така структура додає в лист наступну інформацію:

1. Навчальний предмет
2. Назва матеріалу
3. Назва розділу пов’язаного з кожним навчальним матеріалом
4. Назва кожної теми пов’язаних з кожним навчальним матеріалом
5. Дані про відправника,
6. Додаткова інформація, надана відправником.

Завдяки обраній структурі кожен студент може впорядкувати листи як він забажає і знайти необхідні йому матеріали. Це вкрай важливо тому що легкий доступ до навчальної інформації має один з найвищих пріоритетів.

1.3 Аналіз існуючих рішень

1.3.1 Хмарні сховища

Для поширення навчальних матеріалів потрібне місце де вони будуть зберігатися. Для таких цілей добре підходить хмарні сховища.

Плюси:

1. Надійне зберігання даних
2. Більшість хмарних сховищ надають можливість поширити доступ до електронних ресурсів
3. Невисока ціна. Хмарне зберігання позбавляє від необхідності платити за ліцензії на програмне забезпечення та оновлення, оскільки все це входить в одну глобальну щомісячну ціну. Хмарне зберігання також уникає необхідності вкладати кошти в дорогу серверну інфраструктуру, оскільки хмарна компанія надає вам цей веб-сайт.

4. Масштабованість. Необхідно платити лише за необхідну кількість пам’яті. За необхідності, можна просто розширити обсяг наявного сховища. і навпаки.

Мінуси

1. Немає інструментарію для управління навчальними матеріалами.
2. Немає можливості пов’язати ресурс з робочою навчальною програмою
3. Відсутня можливість розписати план поширення матеріалів
4. Хмарні середовища – виступають третьою особою і не можуть бути використана для зберігання документів, що мають бути захищенні від третіх осіб.
5. Не призначені для взаємодії між викладачем та студентами.

Отже таке рішення окремо не може покрити вимоги які поставлені у цій дипломній роботі, може бути використано як частина системи з централізованого управління розповсюдженням та опрацюванням навчальної літератури.

1.3.2 Електронний кампус НТУУ “КПІ”

КПІ вже має своє рішення під назвою “Електронний кампус НТУУ “КПІ”. Ось деякі причини, чому ЕК КПІ не підходить покриття поставлених цілей у дипломній роботі.

1. У більшій мірі не використовується як інструмент для поширення навчальних матеріалів
2. Це рішення на момент написання диплому не набуло широкого поширення серед студентів
3. Можливість поширювати навчальні матеріали присутня у вузького кола користувачів.
4. Система не має інтуїтивного інтерфейсу
5. Вимагає наявності облікового запису для користування навчальними матеріалами.
6. Немає формату поширення матеріалів через електронну пошту, що значно ускладнює процес поширення матеріалів.

7. Відсутня можливості заздалегідь налаштувати план поширення матеріалів на весь навчальний семестр
8. КПІ орієнтовна система. Однією з цілей цієї дипломної роботи стало створення системи, що орієнтована на всі вищі навчальні заклади, що ЕК КПІ не реалізувати.

1.3.3 Google Classroom

Google Classroom є частиною пакету G Suite for Education, який включає Gmail, Google Drive, Google Calendar та інші додатки. Він орієнтований на викладачів та студентів в школах, так і в області вищої освіти.

Плюси [2]

1. Повторне використання завдань, тестів чи іншого вмісту курсу
2. Можливість додати вміст до завдань, таких як відео, PDF-файли, Google Документи або Google Форми.
3. Переглядайте завдання, оголошення та інші ресурси на сторінці ресурсів класу.

Мінуси

1. Відсутність можливості використовувати власні засоби збереження даних. Деякі дані мають бути захищені від третіх обличь, чого не можна гарантувати використовуючи Google Classroom.
2. Дуже сильний зв’язок з іншими сервісами Google. Перш за все, це значно ускладнює перехід до іншої платформи. Перший раз користувачі Google можуть заплутатися, оскільки є кілька кнопок із значками, знайомими лише користувачам Google [3].

1.3.4 Moodle

Moodle - це платформа управління процесом навчання з відкритим кодом. Ця система була розроблена, з метою допомогти школам навчати своїх учнів. Moodle заснований на модульній конструкції, яка дозволяє вчителям та адміністраторам створювати власну навчальну програму, використовуючи додатки для різних робочих процесів, змісту та заходів [4].

Плюси:

1. Гнучкі варіанти розгортання,

2. викладачі можуть вибирати власний контент та діяльність

3. виділений мобільний додаток Moodle.

Мінуси:

1. Плани Moodle for School вміщують лише до 500 користувачів

2. Висока ціна

1.3.5 Платформи онлайн курсів

Прикладом такої платформи можна назвати Coursera. Курси що пропонує Coursera містить вікторини, щотижневі вправи, однорангові завдання, а іноді і заключний проект чи іспит.

Такі платформи переслідують інші цілі, ніж ті, що поставлені у цій дипломній роботі. Такі платформи добре підходять для самостійного навчання проте вони не підходять для стаціонарного виду навчання. Також необхідно регулярно постійно оновлювати викладений матеріал. Такий процес потребує значних ресурсів, особливо коли мова йде про перезапис відео-, аудіо-матеріалів.

Важливо пам’ятати, що впровадити таку систему в навчальний процес можна, але частково, як допоміжний інструмент, що не можна назвати системою саме з централізованого управління розповсюдженням та опрацюванням навчальної літератури.

1.4 Висновки до розділу

В цьому розділі було проведено огляд предметної області та було поставлено конкретні проблеми, що будуть вирішені за допомогою розроблюваної інформаційної системи. Більш детально розглянуто:

1. проблема вартості впровадження системи в навчальний процес,

2. проблема поширення навчальних матеріалів,

3. проблема відсутності єдиного підходу до оформлення електронних листів.

Було проведено аналіз існуючих рішень. Серед них було звернуто увагу як на конкретні інформаційні системи так і на цілі категорії web-систем з метою виявити їх сильні та слабкі сторони. Проведений аналіз допоможе у проектуванні майбутньої інформаційної системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ WEB-СИСТЕМИ

2.1 Огляд бізнес-ролей та їх можливостей

В розроблюваній web-системі кожен з користувачів може мати декілька бізнес-ролей. Бізнес-роль – це група споріднених навичок із рівнем повноважень направлених на виконання певного завдання.

Для того, щоб розуміти які можливості надає система, у нижче розібрано кожну роль окремо.

2.1.1 Вчитель

Система на дійсній версії ставить підтримку функціональності для вчителя у більш високий пріоритет, ніж для студента. На це є достатньо вагома причина, а саме студенту необов’язково мати окремий клієнт для того, щоб взаємодіяти з навчальними матеріалами. Про це більш докладніше у пункті про роль студента в рамках web-системи.

Вчитель має можливість імпортувати навчальні матеріали. Для імпортованого матеріалу у системі автоматично згенерується посилання за яким можна отримати доступ до файла.

Імпортований ресурс можна переглядати, редагувати або видаляти. Вчитель має можливість зафіксувати авторство, проте це не вплине на збережену інформацію про те, хто і коли додав чи хто останній редагував цей ресурс. Іншими словами система підтримує аудит навчальних матеріалів.

2.1.2 Студент

Одною з характеристик цієї системи це простота впровадження, у перспективі того, що непотрібно організовувати процес інтеграції студентів. Під цим мається на увазі, що кожен студент буде отримувати нові навчальні матеріали на свою електрону пошту.

Отриманий лист з матеріалами, чітко структурований, має визначену тему листа, примітки за який курс, семестр. Чітко визначено до якого навчальної

дисципліни відноситься лист та навчальні матеріали в ньому з вказаним розділом кредитного модуля та конкретною темою.

2.1.3 Адміністратор з керування контентом

Виокремлена роль у якої є всі необхідні права та доступ щоб імпортувати навчальні матеріали до системи, присвоювати кожному ресурсу всю необхідну інформацію. Також адміністратор з керування контентом створює нові облікові записи і за необхідності пов’язує їх з певною групою, чи заповнює іншу інформацію про користувача.

Адміністратор з керування контентом має великий потенціал, адже з зростанні можливостей системи, виникає потреба в управлінні даними. Для таких цілей і була створена ця роль.

2.1.4 Адміністратор web-системи

Для підтримки життєдіяльності системи, було впроваджено окрему бізнес-роль – адміністратор web-системи. Завдяки цьому є можливість надати доступ до технічних характеристик системи, без доступу до “чутливих” даних. Під чутливими даними мається на увазі ті дані, що мають бути приховані від сторонніх осіб, такі як особисті дані студентів чи викладачів.

Отже користувач з такою роллю насамперед має доступ до метрик системи. На рисунку 2.1 зображено приклад того як виглядає вікно з метриками.

Інше вікно, що доступно такому адміністратору вікно стану web-системи. У цьому режимі система звітує про коректність роботи окремих системи.

Для перегляду налаштування різних конфігурацій адміністратор має окремий інструмент. Цей інструмент має зручний фільтр користуючись яких є можливість швидко знайти необхідну інформацію.

Адміністратор також має можливість у гнучкий спосіб керувати логами системи. В логах фіксується дуже багато службової та статистичної інформації. Задля підвищення інформативності цих записів, адміністратор має можливість встановити потрібну фільтрацію

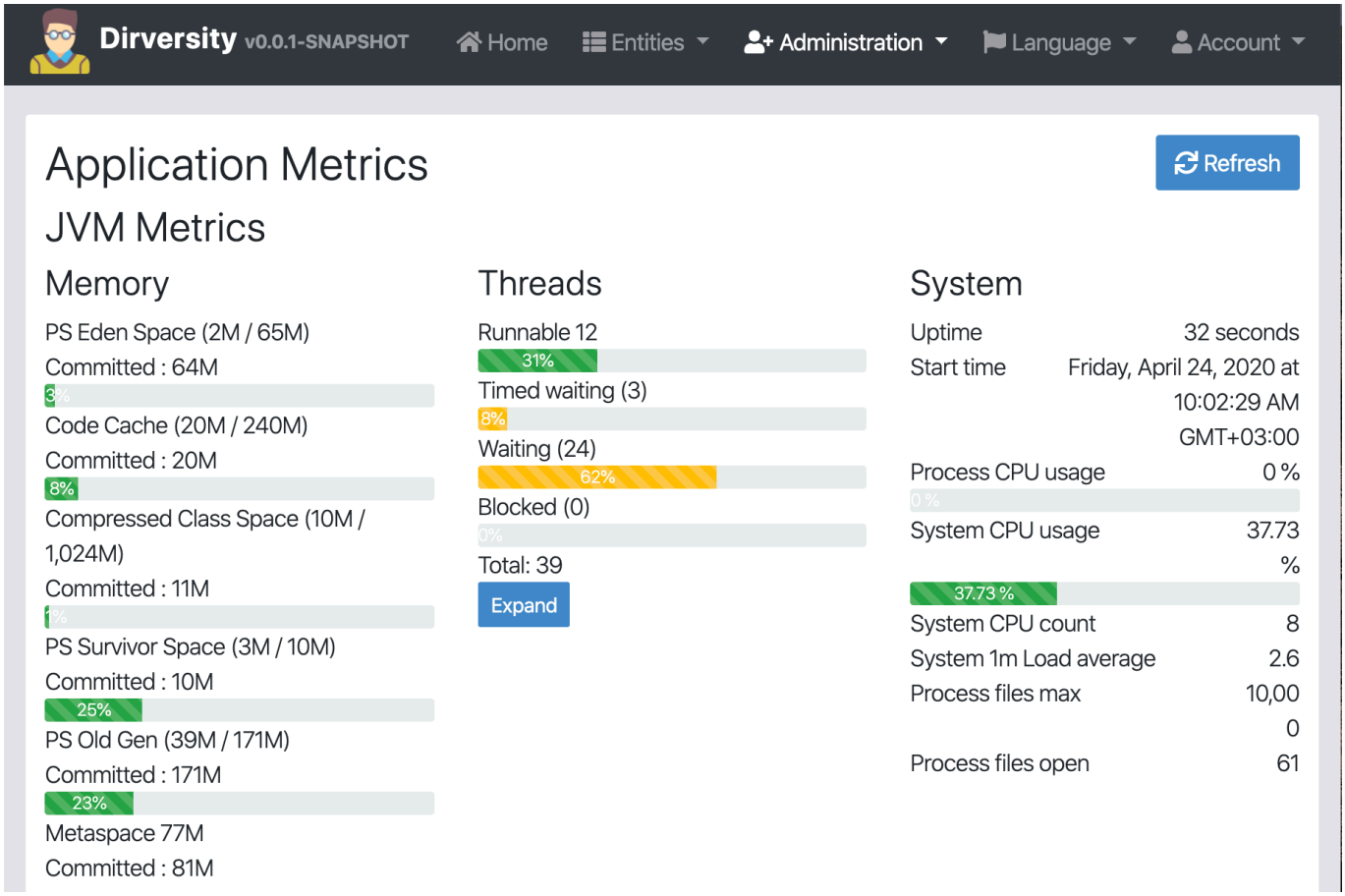

Рисунок 2.1 Вікно метрик web-системи

2.1.5 Адміністратор (супер адміністратор)

Адміністратор має всі ті самі можливості, що і кожна з наведених ролей. Проте є і унікальні можливості. Однією з таких можна виділити повний спектр управління користувачами.

Управління користувачами включає в себе наступні можливості:

1. Реєстрація нового користувача
2. Перегляд інформації про кожного наявного в системі користувача
3. Видалення користувача з системи
4. Блокування користувача у системі. В даному випадку вся наявна інформація про користувача збережеться, проте система заблокує для даного користувача.
5. Редагувати ролі (профілів) користувача в системі.
6. Редагувати інформацію про кожного наявного в системі користувача.

7. Визначення мови інтерфейсу за змочуванням.
8. Перегляд аудиту кожного користувача, а саме інформацію про те хто і коли створив користувача або хто і коли останній раз вносив зміни.

2.2 Проектування зовнішнього вигляду можливостей користувача

Розглянемо окремі можливості системи більш детально. Кожна розглянута функціональність містить розроблений інтерфейс користувача для кращого розуміння очікуваного результату.

2.2.1 Управління навчальними матеріалами

Для перегляду усіх наявних навчальних матеріалів необхідно перейти на сторінку “Ресурси” в розділі “Перейти до” у верхній панелі вікна.

Важливо зауважити, що якщо користувач немає прав адміністратора, система наддасть доступ тільки до тих ресурсів, що були створені саме користувачем, інакше повернуться усі наявні ресурси. На сторінці загального перегляду (рисунок 2.2) ресурсів знаходиться інформація про індифікаційний код ресурсу, авторів та його назви. Також на цій сторінці можна побачити дані, що стосуються аудиту, а саме:

- хто створив ресурс,
- дата створення ресурсу,
- ким останній раз було змінено ресурс,
- коли останній раз було змінено ресурс.

Важливо зауважити, що ні в кого немає можливості редагувати дані аудиту, вони керуються системою в автоматичному режиму. У разі якщо, щось було створено саме системою, то на такий випадок буде використано ім’я “system”. На цій самій сторінці є можливість перейти до створення нового ресурсу, перегляду вже створеного, його редагуванню чи видаленню.

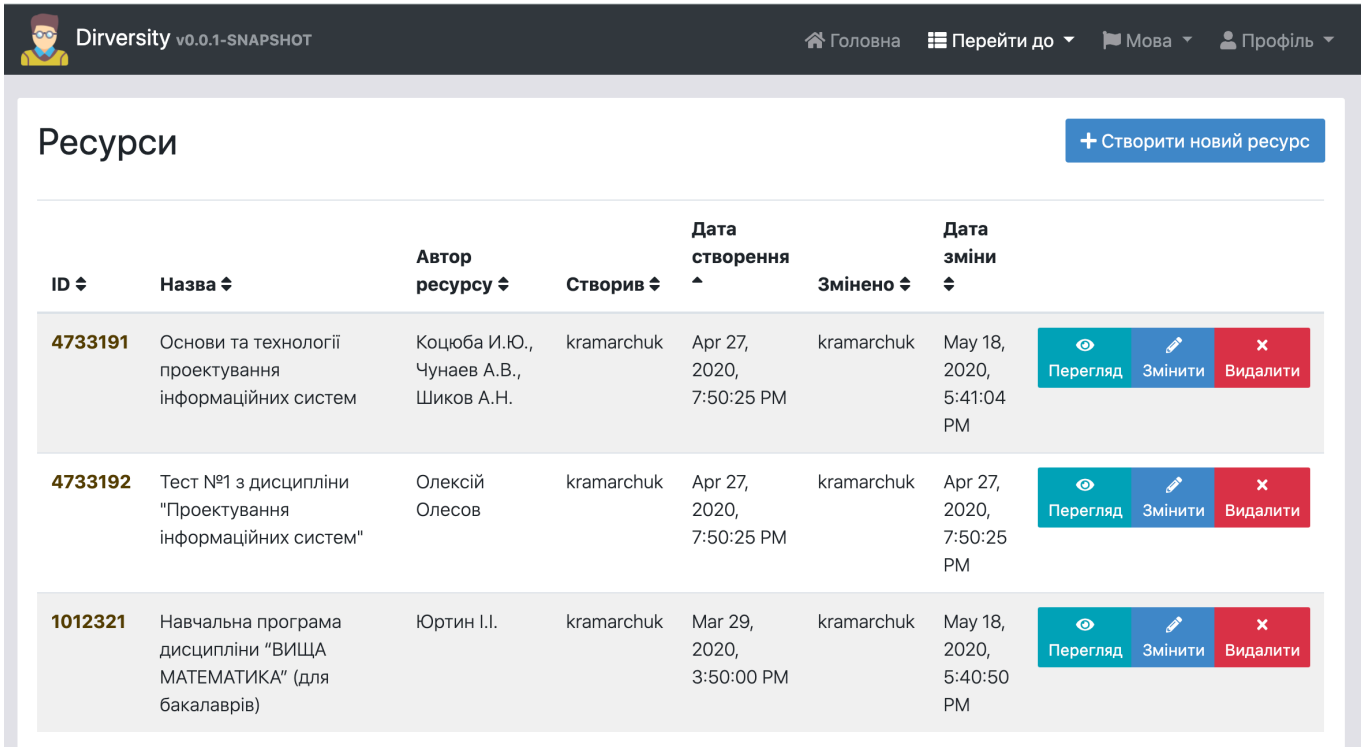

Рисунок 2.2 Сторінка перегляду ресурсів.

При переході у режим детального перегляду ресурсу потрапляємо на сторінку зображену на рисунку 2.3. Ця сторінка має всі дані, щ обули описані вище і декілька додаткових, а саме:

- тип ресурсного файла,
- URL адреса ресурсного файлу,
- ідентифікатор ресурсного файлу,
- теми пов'язані з ресурсом.

При створенні чи редагуванні ресурсу в користувача є можливість завантажити відповідний електронний файл. Після збереження ресурсу система опрацьовує електронний файл і робить наступні дії:

1. Відправляє файл у сховище з іншими ресурсами.
2. Визначає для цього файла права на перегляд
3. Генерує ідентифікатор
4. Генерує посилання на цей файл для поширення у майбутньому.
5. Зберігає всі згенеровані дані до системи.

Користуючись отриманими даними система на сторінці перегляду окремого ресурсу додає на панель кнопку “Перейти до ресурсу”. Коли користувач натискає на неї у новому вікні відкривається сам електронний файл. Також спираючись на ці система автоматично прикріплює ресурси до електронних листів, що дає можливість поширити ці матеріали.

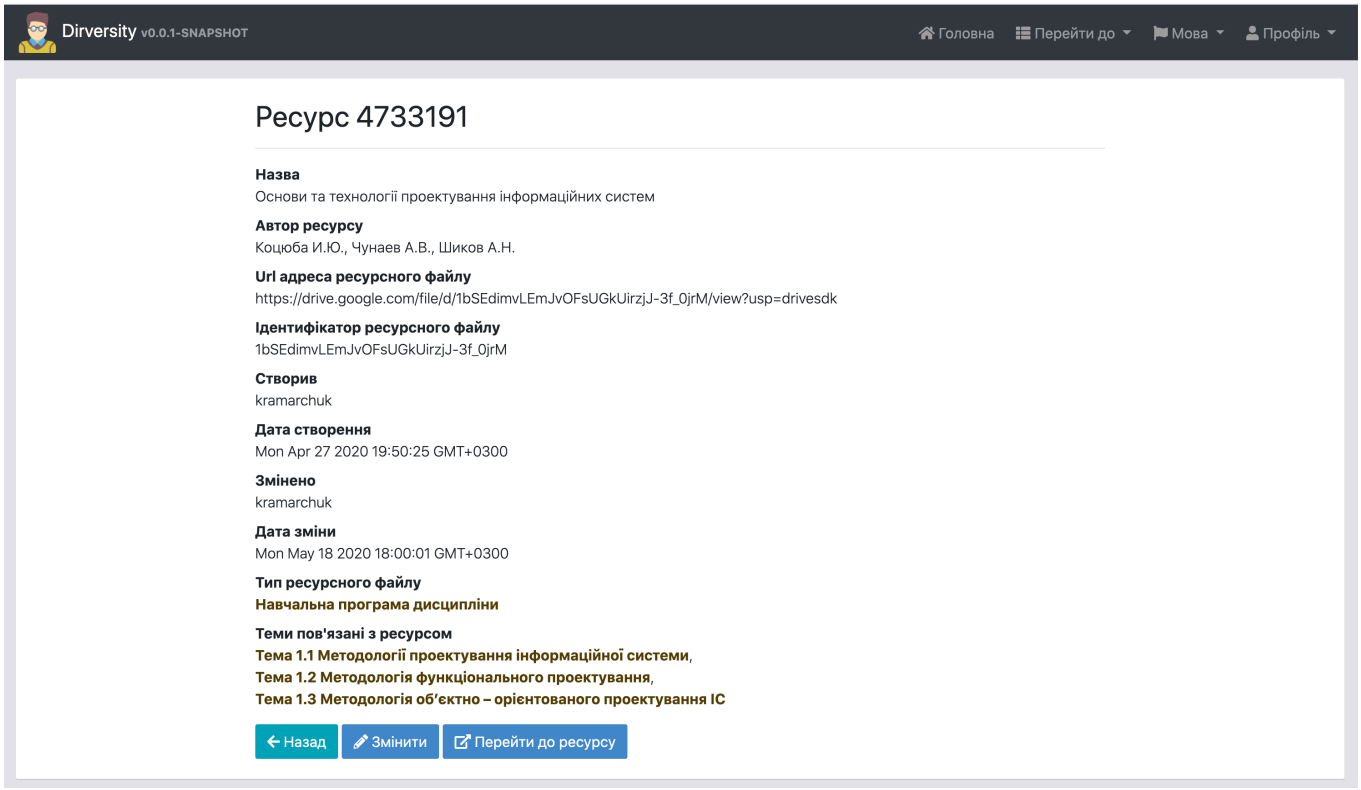

Рисунок 2.3 Режим перегляду окремого ресурсу

2.2.2 Робота з робочими навчальними програмами

Для того, щоб розпочати роботу з робочими навчальними програмами необхідно перейти на відповідну сторінку (рисунок 2.4) в розділі “Перейти до” у верхній панелі вікна.

На сторінці перегляду усіх робочих начальних програм є можливість перейти до перегляду ресурсу, його редагуванню чи видаленню. Якщо натиснути на назву оригінального файлу система відкриє сторінку відповідного ресурсу, що матиме вигляд, як на сторінці зображеної на рисунку 2.3.

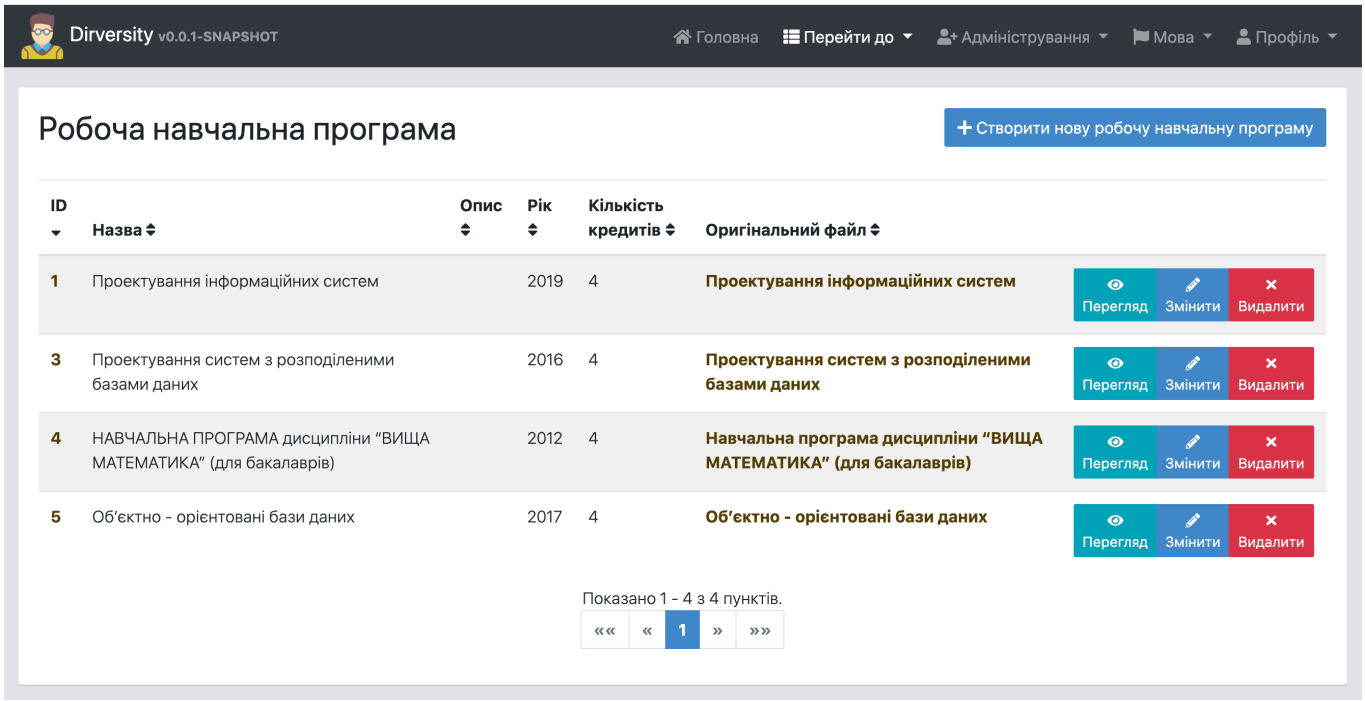

Рисунок 2.4 Сторінка перегляду робочих навчальних програм

Кожна навчальна програма може містити в собі посилання на оригінальний файл, що представляє собою ресурс. Цей ресурс містить в собі електронний файл, завантажений в сховище ресурсів і має згенероване посилання на цей ресурс.

Також кожен з навчальних програм у режимі перегляду (рисунок 2.5) надає можливість переглянути наступні дані:

- назва
- рік створення,
- пояснювальна записка,
- посилання на оригінальний файл,
- теги – мета-дані, що дають більш широке уявлення про зміст начальної програми,
- автори,
- список розділів.

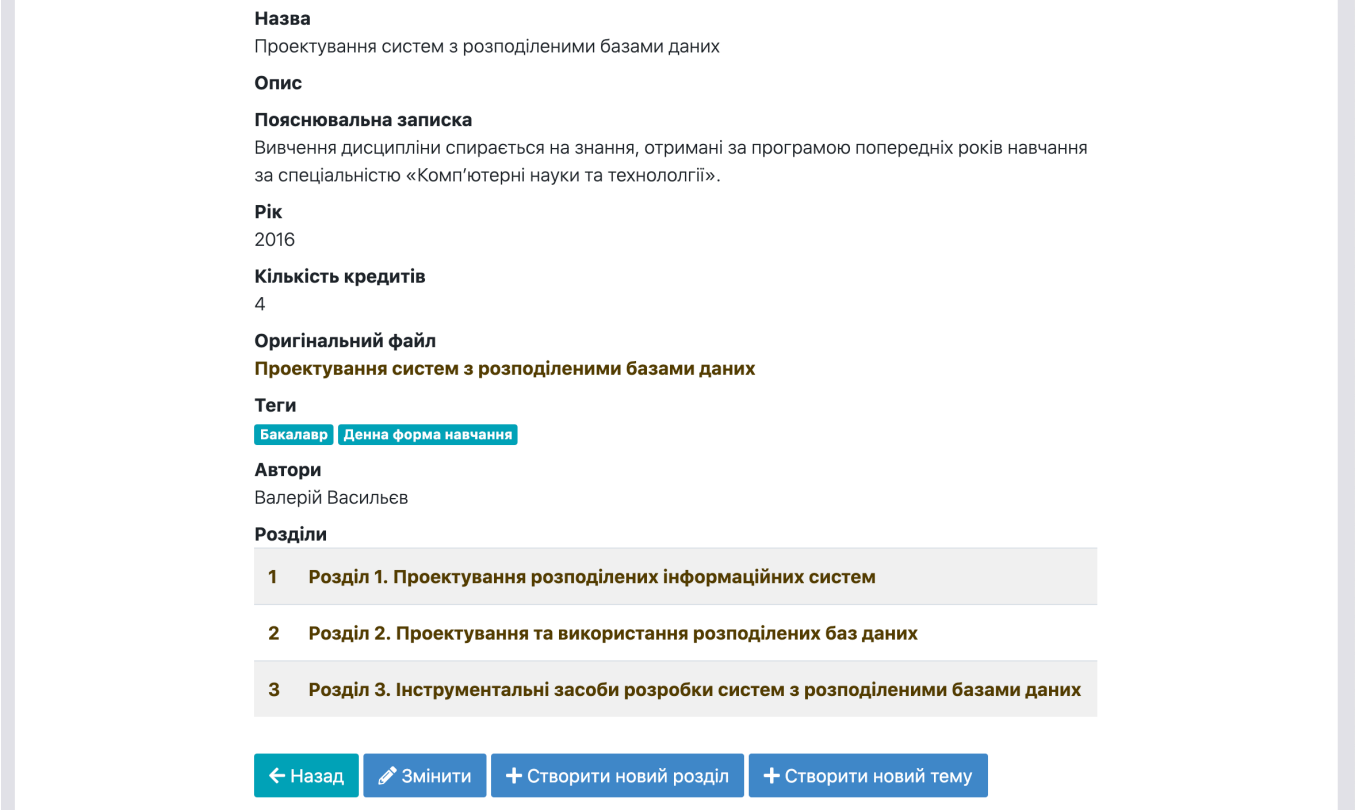

Рисунок 2.5 Режим перегляду робочої навчальної програми

У режимі перегляду робочої навчальної програми можна створити як новий розділ так і нову тему. Список розділів, що надається з кожною навчальною програмою може містить у собі посилання на сторінки детального перегляду цих розділів. Приклад такої сторінки зображено на рисунку 2.6.

Сам розділ містить список тем, які також є посиланнями на сторінки детального перегляду тем. На теми можна посилатися при відправці електронного листа у результаті, чого студенти матимуть докладну інформацію про те, чого саме стосуються отримані навчальні матеріали.

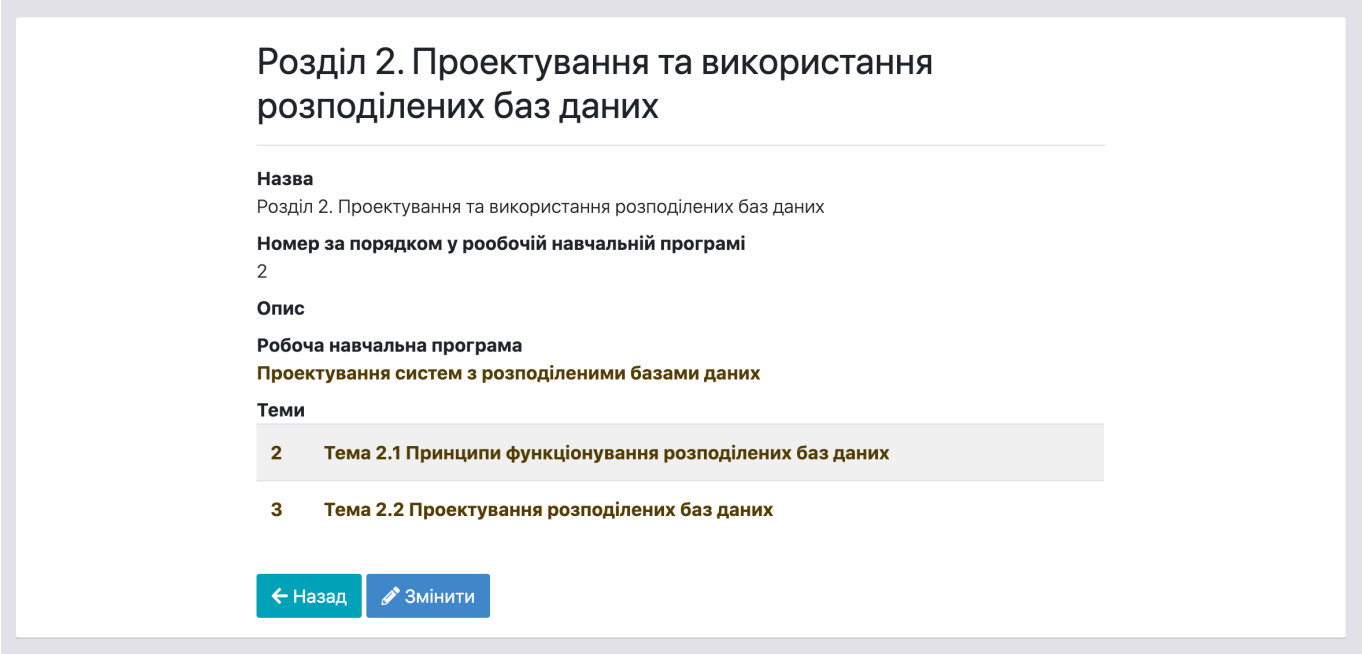

Рисунок 2.6 Режим перегляду розділів робочої навчальної програми

2.2.2 Поширення навчальних матеріалів

Для того, щоб поширити навчальні матеріали необхідно створити електронний лист або скористатись існуючим. Для цього користувач може перейти на відповідну сторінку “Електронні листи” (рисунок 2.7) в розділі “Перейти до” у верхній панелі вікна. Необхідно підкреслити той факт, що для користувача, що в даний момент авторизований в системі система наддасть доступ тільки для тих електронних листів, що були створені саме цим користувачем.

На сторінці загального перегляду електронних листів знаходиться інформація про ідентифікаційний код ресурсу, авторів та його назви. Також на цій сторінці можна побачити дані, що стосуються аудиту, такі самі які має ресурс, а саме:

- хто створив електронний лист,
- дата створення електронного листа,
- ким останній раз було змінено електронний лист,
- коли останній раз було змінено електронний лист.

Редагувати дані аудиту електронних листів неможливо, вони так само як і аудит ресурсів керуються системою в автоматично. Якщо система сама згенерує

електронний лист, користувачем буде виступати користувач з іменем “system”. Детальніше про аудит у розділі 4.

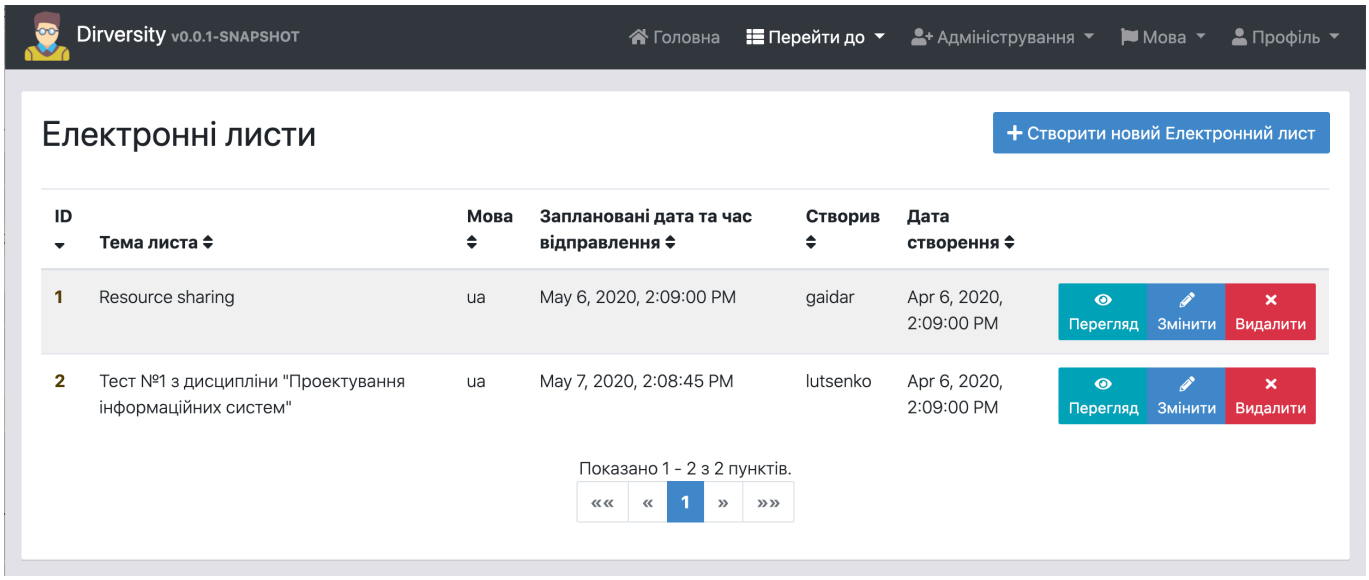

Рисунок 2.7 Сторінка перегляду електронних листів

Також можна помітити, що в режимі перегляду усіх електронних листів окрім ідентифікаторів і аудиту розміщено додаткові атрибути, а саме:

- Тема листа. Це поле було розміщено саме у цьому режимі для більшої інформативності.
- Мова. Коли лист генерується перед відправкою, система бере до уваги яку мову слід використовувати.
- Заплановані дата та час відправлення. Цей атрибут визначає коли електронний лист має бути відправлений. Таким чином можна заздалегідь визначити дату та час поширення для всіх навчальних матеріалів за весь семестр.

При переході до режиму детального перегляду електронного листа (рисунок 2.8) або його редагування користувач спостерігає ряд нових полів:

- Тіло листа
- Кому (To). Це поле необхідно для поодичного визначення отримувачів.
- Яким групам відправити. В цьому полі користувач має можливість визначити цілі групи користувачів, які мають отримати електронний лист.

- Копія (Cc). В цьому полі поодинично визначається кого з користувачів додати до копії листа.
- Які групи додати до копії. В цьому полі користувач має можливість визначати цілі групи користувачів, яких необхідно додати до копії.
- Прикріплені ресурси. Це поле необхідно для того щоб прикріпити відповідні ресурси до листа.

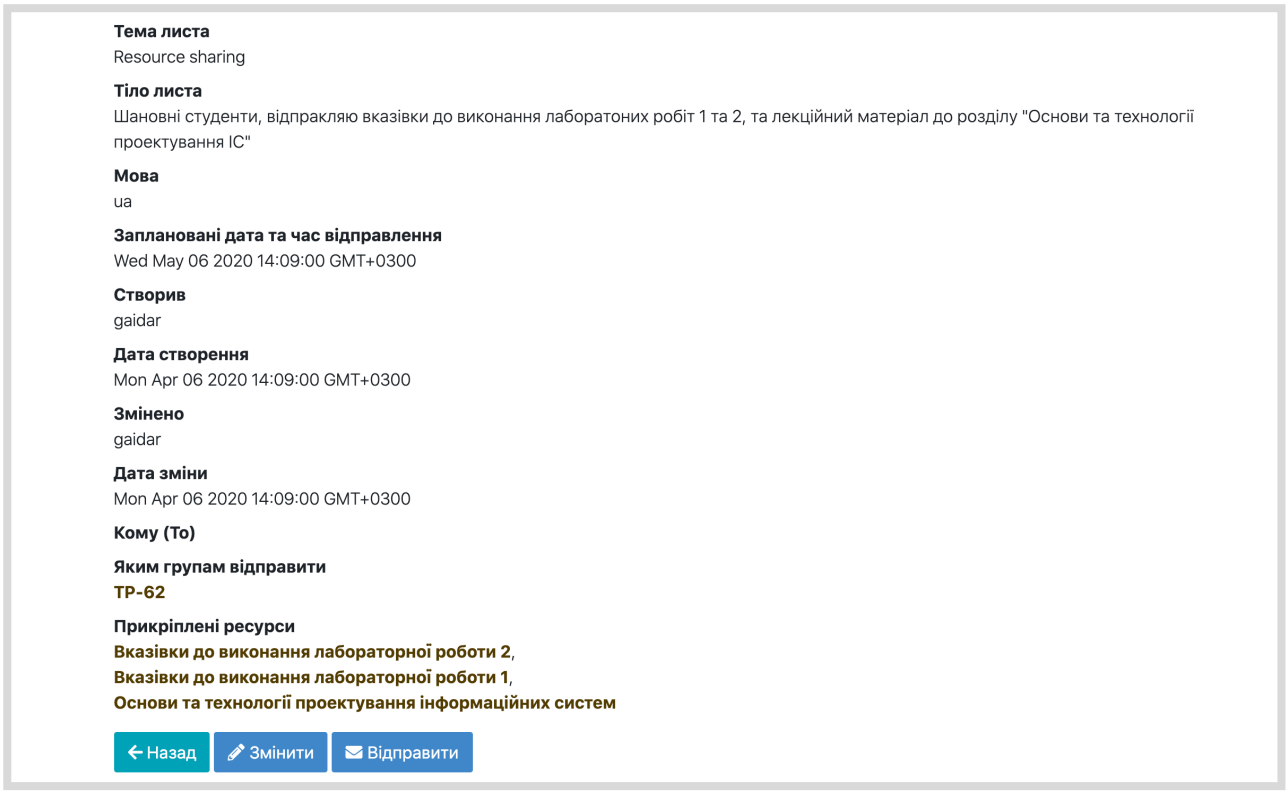

Рисунок 2.8 Режим детального перегляду електронного листа

Після створення листа у користувача є можливість натиснути у нижній панелі на кнопку “Відправити”, після чого система базуючись на всіх атрибутах згенерує електронний лист і відправить його користувачам згідно встановлених правил.

У випадку коли встановлено дата та час відправлення електронного листа, система користуючись внутрішніми механізмами відправить лист у потрібний день та час. Детальніше про цей процес у розділі 4.

2.4 Висновки до розділу

У цьому розділі задля розуміння можливостей системи було проведено аналіз кожної з бізнес-ролей. Серед них розглянуто наступні ролі:

1. вчитель,
2. студент,
3. адміністратор з керування контентом,
4. адміністратор web-системи,
5. адміністратор (супер адміністратор).

Також було розглянемо окремі можливості системи більш детально, адже деякі з них є незалежними від ролі користувача у системі. У кожній такій функціональності розглянуто те, який зовнішній вигляд матиме розроблювана інформаційна система, задля кращого розуміння очікуваного результату.

Було детально розглянуто управління навчальними матеріалами, як створювати нові ресурси, як перегляди чи редагувати вже створені або навіть видаляти. Освітлено повну структуру сутності ресурсів у системі. Описано як в системі працює аудит, та яку інформацію можна отримати з нього.

У цьому розділі описано концепцію управління робочими навчальними програмами в рамках розроблюваної інформаційної системи. Описано яка внутрішня структура присутня в сутності робочих навчальних програм, якими атрибутами вона володіє та що означає кожен з них.

Система надає можливість поширювати ресурси через електронні листи. Для створення таких листів розроблювана система надає відповідні інструменти про користування якими детально розписано в цьому розділі. Окрему увагу приділено можливості відправлення електронних листів як вручну, натиснувши на відповідну кнопку, так і автоматично у заплановані дату та час, користуючись внутрішніми механізмами web-системи.

РОЗДІЛ 3
АРХІТЕКТУРА WEB-СИСТЕМИ

3.1 Аналіз існуючих архітектурних підходів

Вибір архітектури програмного комплексу завжди є одним з найважливіших питань під час створення інформаційної системи. Як свідчить досвід розробки, добре спроектована архітектура може зекономити багато часу та ресурсів, натомість погана, може стати фатальною навіть для проекту з неймовірно гарною ідеєю. Ось деякі з причин [5], що можуть виникнути при погано спроектованій архітектурі:

- погана зручність використання,
- погана продуктивність,
- порушення безпеки,
- погана масштабованість,
- погана ремонтопридатність,
- не працює належним чином,
- не відповідає функціональності та вимогам користувачів.

Постає завдання до аналізу наявних підходів до проектування архітектури інформаційних систем та виявлення найкращого для створення системи управління розповсюдженням та опрацюванням навчальної літератури.

Архітектура програмного забезпечення стосується прийняття набору стратегічних технічних рішень, пов'язаних із програмним продуктом, документування їх та забезпечення їх реалізації. Проектування архітектури стала однією з найважливіших етапів у побудові програмного забезпечення, особливо у царині розвитку високонавантажених систем.

Дуже важливо розуміти та провести оцінку переваг та недоліків тієї чи іншої архітектури. Для розроблюваної системи управління дуже важливо мати можливість, з мінімальною витратою ресурсів, забезпечити інтегрованість системи до різноманітних сховищ даних. З урахуванням цього буде проходити подальший аналіз.

Два найпоширеніших види архітектур є монолітна та мікросервісна. Впродовж аналізу буде розглянутий, ще один підхід, але спочатку буде розглянуто перші два.

3.1.1 Монолітний архітектурний стиль

До стрімкого розвитку концепції мікросервісів, більшість веб-додатків були побудовані з використанням монолітного архітектурного стилю. У монолітній архітектурі додаток поставляється як єдиний розгорнутий програмний артефакт [6]. Весь інтерфейс користувача, бізнес логіка та рівень доступу до бази даних упаковуються разом в один артефакт програми та розгортаються на серверу.

Монолітна архітектура вважається традиційним способом побудови додатків. Монолітний додаток будується як єдине і неподільне ціле. Він уніфікований і всі функції керуються та обслуговуються в одному місці.

Зазвичай монолітні програми мають одну велику базу коду та незначну модульність. Якщо розробники хочуть щось оновити або змінити, вони отримують доступ до тієї ж бази коду. Тому, вони вносять зміни в одне місце одночасно.

На рисунку 3.1 зображено загальний вигляд монолітного архітектурного стилю. При такому стилі програмні компоненті, які реалізують бізнес-логіку, інтерфейс користувача та контроль рівня доступу до даних знаходяться в рамках єдиного модуля. На схемі зображена лише одна база даних, проте їх може бути більше, що разом з зростаючою бізнес-логікою буде ускладнювати подальше впровадження нової функціональності.

Сильні сторони монолітної архітектури:

1. Простіший процес налагодження та тестування. На відміну від архітектури мікросервісів, монолітні програми набагато простіше налагоджувати та перевіряти. Тестування може бути запущено набагато швидше, оскільки монолітний додаток є єдиною нероздільною одиницею.
2. Менше наскрізних проблем (проблеми, які впливають на всю програму, такі як реєстрація, обробка, кешування та моніторинг продуктивності). У монолітній програмі ця область функціональності стосується лише одного додатка, тому з ним легше впоратися.

3. Простий у розвитку. Так як монолітний підхід є стандартним способом побудови додатків, будь-яка інженерна команда має правильні знання та можливості для розробки монолітного додатку.
4. Простий у розгортанні. Простота монолітних додатків дає ще одну перевагу - легке розгортання програмного артефакту. Завдяки простій структурі монолітних додатків, немає потреби обробляти багато розгортань - лише один файл або каталог.

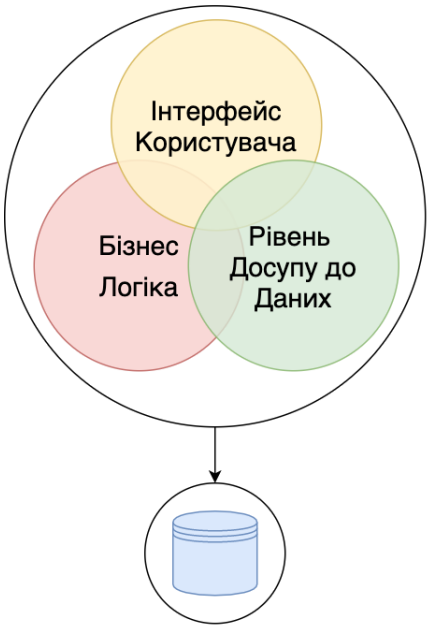

Рис. 3.1 Монолітна архітектура

Слабкі сторони монолітної архітектури:

1. Коли масштабування монолітного додатка стає занадто складним, щоб зрозуміти, стає важко керувати такою комплексною системою коду в межах однієї програми.
2. Внесення змін. Важко здійснити зміни в такій великій і складній системі з дуже щільним з'єднанням. Будь-яка зміна коду впливає на всю систему, тому її потрібно ретельно координувати. Це робить процес загального розвитку набагато довшим.

3. Неможливо самостійно масштабувати окремі компоненти, лише всю програму одночасно.
4. Нові технологічні бар’єри. Застосувати нову технологію в монолітній програмі вкрай проблематично, оскільки тоді всю програму потрібно переписати.

Після аналізу можна виокремити наступні причини які вказують на те, що слід використати монолітну архітектуру для створення системи управління розповсюдженням та опрацюванням навчальної літератури.

1. Невелика команда на етапі заснування проекту.
2. Проект є підтвердженням концепції. У цьому моноліт ідеально підходить для швидкої ітерації продукт, що дозволить швидко отримати результати.
3. В команді немає DevOps інженера.

3.1.2 Мікросервісний архітектурний стиль

Як сказав автор багатьох книг і статей з архітектури програмного забезпечення Мартін Фаулер: “Мікросервісний архітектурний стиль - це підхід до розробки єдиного додатку як набору невеликих сервісів, кожен із яких працює у своєму процесі та спілкується з легкими механізмами, часто API HTTP-ресурсу” [7].

У той час як монолітна програма є єдиною одиницею, архітектура мікросервісів розбиває її на набір менших незалежних одиниць. Кожен мікросервіс - це невеликий додаток, який має власну архітектуру, що складається з бізнес-логіки разом з різними адаптерами.

На рисунку 3.2 зображено загальну схему додатку розробленого з використанням мікросервісної архітектури.

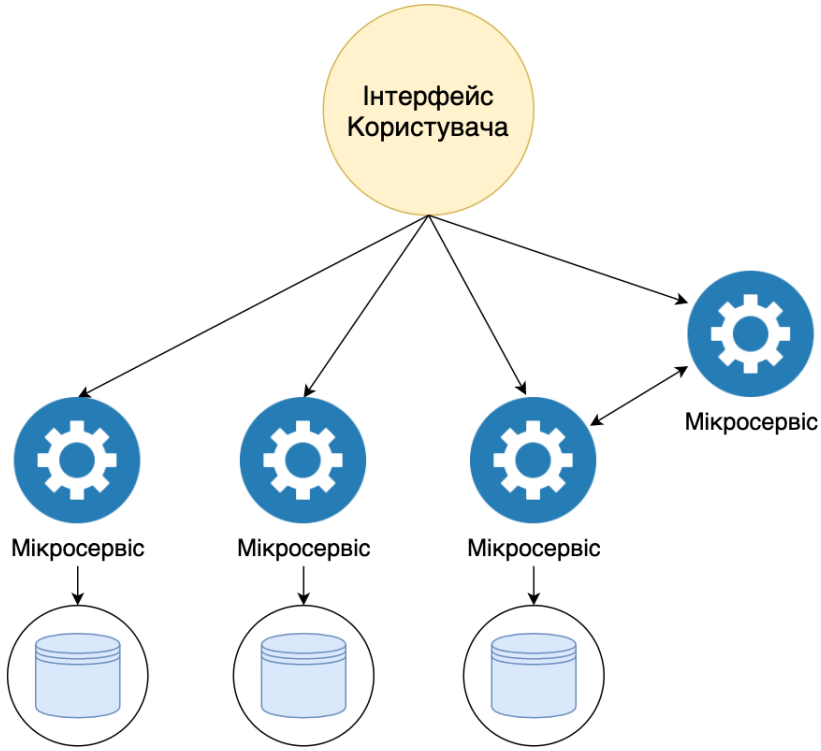

Рисунок 3.2 Мікросервісна архітектура

Є дуже багато визначень мікросервісної архітектури, тим не менш більшість з них присвоюють даному архітектурному стилю наступні характеристики:

1. Логіка застосування розбивається на дрібнозернисті компоненти з чітко визначеними межами відповідальності.
2. Мікросервіси спілкуються на основі декількох основних принципів і використовують легкі протоколи зв'язку, такі як HTTP та JSON для обміну даними між сервісами споживачем та постачальником.
3. Кожен компонент має невелику область відповідальності і розгортається повністю незалежно один від одного. Мікросервіси повинні нести відповідальність за окрему частину бізнес-логіки. Також мікросервіс повинен бути багаторазовим для використання.
4. Основна технічна реалізація послуги не має значення, оскільки програми завжди спілкуються з нейтральним технологічним протоколом (наприклад XML, JSON, Protobuf). Це означає, що додаток, побудований за допомогою мікросервісного додатку, може бути побудований за допомогою декількох мов та технологій.

5. Мікросервіси - за їх малим, незалежним та розподіленим характером дозволяють організаціям мати невеликі команди розвитку з чітко визначеними зонами відповідальності.

Сильні сторони мікросервісної архітектури:

1. Незалежні компоненти. Всі сервіси можна розгорнути та оновити окремо, що дає більшу гнучкість.
2. Помилка в одному мікросервісі впливає лише на певний сервіс і не впливає на всю програму. Крім того, набагато простіше додавати нову функціональність до додатку, що базується на мікросервісній архітектурі, ніж на монолітній.
3. Розділений на більш дрібні та прості компоненти додаток простіше розуміти та керувати ним. Виникає потреба концентруєтесь лише на певній службі, яка пов’язана з цільовою метою.
4. Краща масштабованість. Ще одна перевага мікросервісного підходу полягає в тому, що кожен елемент можна масштабувати незалежно. Таким чином, весь процес є більш економічним, ніж з монолітами, коли всю програму доводиться масштабувати, навіть якщо в цьому немає потреби. У свою чергу кожен моноліт має обмеження щодо масштабованості, тому чим більше користувачів користується додатком, тим більше проблем.
5. Гнучкість у виборі технології. Інженерні колективи не обмежені технологією, обраною з самого початку. Для кожного мікросервісу може бути застосовувані різні технології та підходи.
6. Будь-яка несправність в окремому мікросервісі впливає лише на певну послугу, а не на ціле рішення. Отже всі зміни та експерименти реалізуються з меншими ризиками та меншою кількістю помилок.

Слабкі сторони мікросервісної архітектури:

1. Додаткова складність. Оскільки мікросервісна архітектура є розподіленою системою, ви повинні вибрати та встановити з’єднання між усіма модулями та базами даних.

2. Системний розподіл. Архітектура мікросервісів - це складна система з декількох модулів та баз даних, тому з усіма з’єднаннями потрібно звертатися обережно.
3. Наскрізні проблеми. Створюючи мікросервісний додаток, виникає необхідність у вирішенні наскрізних проблем. Вони включають зовнішню конфігурацію, ведення логів, показники, перевірки стану здоров’я та інші.
4. Тестування. Безліч незалежно розгорнутих компонентів значно ускладнює тестування наскрізної функціональності.

Після порівняльного аналізу описаних варіантів архітектури аналізу зроблено висновок про те, що для створення системи управління розповсюдженням та опрацюванням навчальної літератури слід використати мікросервісну архітектуру, тому що у цьому випадку можна виокремити мікросервіси, що будуть займатися виключно обробкою та збереженням навчальних матеріалів. Такі мікросервіси можуть бути створені незалежним розробником використовуючи будь-які мови програмування (наприклад C++) навіть якщо решта платформи написана за допомогою Java.

Наведена технологія дозволяє, за необхідністю, виконувати заміну мікросервісів.

На рисунку 3.3 зображено процес заміни одного мікросервісу на інший:

1. Розроблюється новий мікросервіс під назвою “Microservice 3”, що замінить старий “Microservice 2”;
2. У регістрі прибираємо інформацію про старий мікросервіс (видаляємо зв’язок A1) і додаємо відомості про новий мікросервіс (додаємо зв’язок B1);
3. У результаті шлюз (Gateway) буде знати що потрібно звертатися вже до нового мікросервісу, тобто замість зв’язку A2 буде використовувати B2.

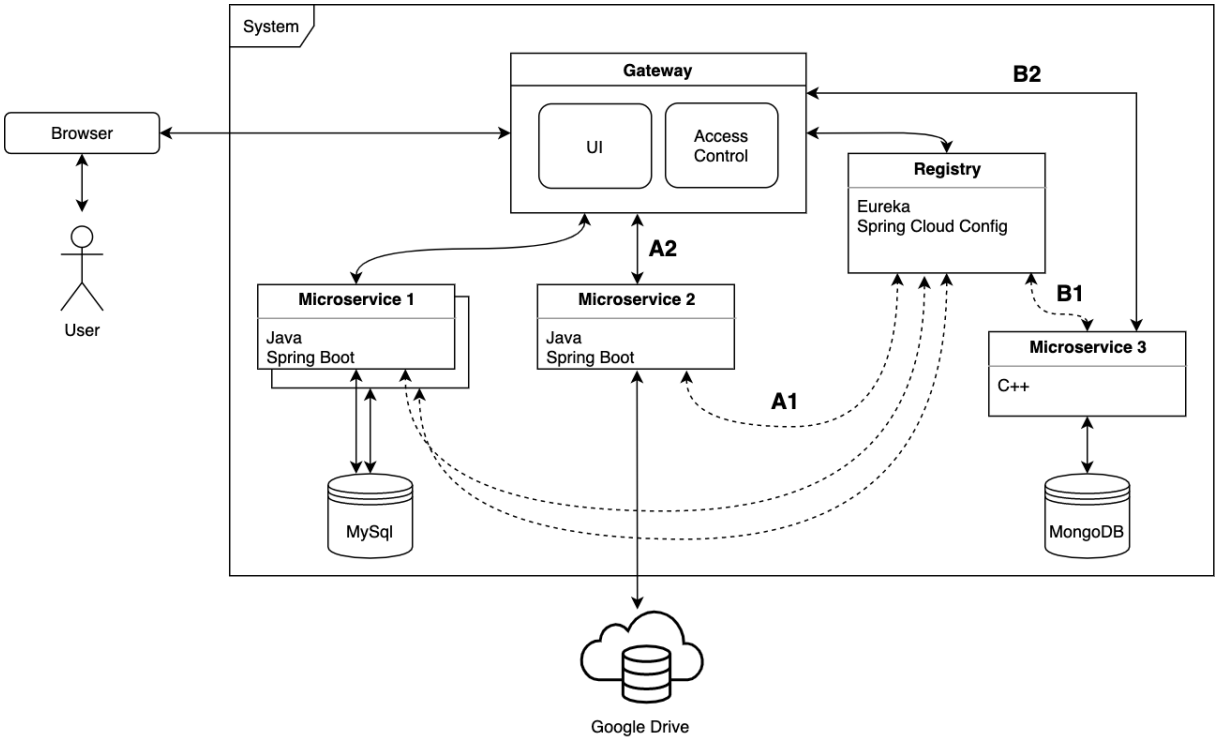

Рисунок 3.3 Заміна одно мікросервісу на інший

3.1.3 Компромісний підхід “Monolith First”

Мікросервісна архітектура має важливе значення для гнучкої розробки, але використання їх має значну перевагу лише для більш складних систем. Управління набором сервісів, значно сповільнить розробку, що говорить про перевагу використання моноліту для більш простих систем [8]. Це призводить до вагомого аргументу стратегії, що починати розробку варто використовуючи монолітний архітектурний стиль, навіть якщо згодом проект отримає користь від архітектури мікросервісів. У такому випадку моноліт слід розробляти ретельно, звертаючи увагу на модульність у програмному забезпеченні, як на кордонах API, так і на збереження даних. При збережені модульності всередині моноліту, перехід до мікросервісної архітектури є відносно простою операцією.

Отже такий підхід, відомій під назвою “Monolith First”, дозволить використати всі переваги монолітної архітектури на початку розробки системи. Наступна перевага проявляється коли з’явиться вимога до використання сторонньої технології для зберігання та опрацювання навчальної літератури. У такому випадку підхід дозволить швидко перейти від моноліту до мікросервісної архітектури.

3.2 Зв’язність між модулями системи

Використовуючи підхід Monolith First насамперед постає питання про слабку зв’язність між модулями. У такому випадку необхідно для цього буде використано Restful API для зв’язку зовні та всередині системи.

REST - це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб. Веб-сервіси, які відповідають архітектурному стилю REST, називаються послугами RESTful Web-сервіси [9].

API - це програмний посередник, який дозволяє двом програмам спілкуватися один з одним [10]. Іншими словами, API - доставляє запит постачальнику, а потім доставляє відповідь.

RESTful веб-сервіси реалізують свій API (RESTful API). Такий API надає доступ до ресурсів за допомогою заздалегідь визначеного набору операцій. Інші види веб-служб, такі як веб-сервіси SOAP, дозволяють власні довільні набори операцій.

Веб-сервіс може мати 2 види API:

1. Відкритий API – такий API може бути використаний, як всередині системи (іншими веб-сервісами) так і іншими клієнтами поза системи. У такому випадку клієнт, що створений для роботи з цією web-системою також є чужорідним.
2. Закритий API – використовується для зв’язку між окремими модулями системи, такий API не може бути використаний іншими третіми особами (third party).

Для отримання інформації через RESTful API немає чіткого стандарту серіалізації даних. Нижче приведено перелік найпопулярніших методів:

1. Extensible Markup Language (XML),
2. JavaScript Object Notation (JSON),
3. Protocol Buffers (Protobuf).

В розроблюваній інформаційній системі буде використовуватись JSON формат.

3.3 SOLID

Відповідно до обраної архітектури, постає підтримувати якість розроблюваної системи. Для цього система має відповідати принципам SOLID:

1. принцип єдиного обов'язку,
2. принцип відкритості/закритості,
3. принцип підстановки Лісков,
4. принцип розділення інтерфейсу,
5. принцип інверсії залежностей.

Принципи SOLID – це стандарт кодування, згідно з яким всі розробники повинні мати чітку концепцію правильного розробки програмного забезпечення, щоб уникнути поганого архітектурного дизайну. При правильному застосуванні це робить код більш масштабованим, логічним та легшим для сприйняття.

Коли програмне забезпечення створюється на основі поганого дизайну, код стає негнучким та нечитабельним. В такому коді навіть невеликі зміни можуть призвести до значних помилок. Беручи до уваги усі принципи SOLID, таких ситуацій можна уникнути.

Ідея застосування принципів у програмах полягає у правильному використанні об'єктно-орієнтованої парадигми, уникаючи таких проблем, як відсутність стандартизації коду чи дублювання.

Для чіткого розуміння того що означають ці принципи, розглянемо кожен з них.

3.3.1 Принцип єдиного обов'язку

Принцип єдиної відповідальності (Single Responsibility Principle) означає, що кожен клас або модуль повинен мати одну і лише одну причину для зміни. Цей принцип дає нам визначення відповідальності, і рекомендації щодо розміру класу. Класи повинні мати одну відповідальність, що і може бути тією єдиною причиною для зміни [11].

Те що один клас повинен служити лише одній цілі не означає, що кожен клас повинен мати лише один метод. Це означає, що такі методи мають стосуватися безпосередньо відповідальності класу. Усі методи та властивості повинні працювати

на одну мету. Коли клас виконує кілька цілей або обов’язків, його відділити в окремий клас.

3.3.2 Принцип відкритості/закритості

Принцип відкритості/закритості (Open/closed principle) спонукає до того, що програмні об’єкти повинні бути відкритими для розширення, проте закритими для модифікації. Цей принцип відсилається до однієї з концепцій ООП, а саме до поліморфізму. Для принципу відкритості/закритості використовують успадкування або реалізують інтерфейси, які дозволяють класам поліморфно підміняти один одного.

Загальна ідея цього принципу говорить про те, що код має бути написаний таким чином, щоб не довелося робити зміни існуючого коду задля додавання нового функціоналу. Ігнорування цього принципу може призвести до ситуації, коли зміна одного з класів вимагає адаптування всіх залежних класів.

3.3.3 Принцип підстановки Лісков

Принцип підстановки Лісков (Liskov substitution principle) спонукає до того, що програмні об'єкти можуть бути заміненими їх нащадками без зміни коду програми. Сама Лісков сформулювала у своїй статті [12] цей принцип наступним чином:

“Нехай q(x) є властивістю правильною для об'єктів x деякого типу T. Тоді q(y) також має бути правильним для об'єктів y типу S, де S — підтип типу T.”

Все це говорить про те, що кожен підклас або похідний клас має можливість замінити свій базового клас.

3.3.4 Принцип розділення інтерфейсу

Принцип розділення інтерфейсу (Interface segregation principle) говорить про те, що клієнт ніколи не має бути змушений реалізовувати інтерфейс, який він не використовує. Інакше його можна було б сформулювати як клієнтів не слід змушувати залежати від методів, які вони не використовують.

Цей принцип застерігає від додавання функціональності у існуючий інтерфейс за допомогою додавання нових методів, натомість він спонукає до створення нового

інтерфейсу у результаті чого, клас, якщо потрібно, може впровадити декілька інтерфейсів одночасно.

3.3.5 Принцип інверсії залежностей

У програмуванні принцип інверсії залежностей (Dependency inversion principle) є способом роз'єднання програмних модулів. Цей принцип стверджує, що

- Модулі високого рівня не повинні залежати від модулів нижнього рівня. Обидва повинні залежати від абстракцій.
- Абстракції не повинні залежати від реалізацій. Реалізації повинні залежати від абстракцій.

Для дотримання цього принципу необхідно використовувати схему дизайну інверсія залежностей. Ін'єкція залежностей стала найпопулярнішим методом вирішення цієї задачі. Найпоширеніший метод ін'єкції залежностей виконується за допомогою конструктора класу.

Найкращий спосіб уникнути небажаного зв'язку між класами ієрархій різного рівня – це використання абстракцій. Кожен з класів на різних рівнях повинен використовувати абстрактний інтерфейс, реалізований класом більш низького рівня.

Таким чином, класи вищого рівня не залежать безпосередньо від класів нижчого рівня, але саме класи нижнього рівня залежать від абстрактних інтерфейсів, оголошених у класах верхнього рівня та можуть викликатися ними.

3.4 RESTful API

Задля дотримання високої якості написання API було вирішено дотримуватись архітектурного стилю REST (Representational State Transfer). Цей підхід був розроблений спеціально для мережевих протоколів, що забезпечують доступ до різноманітних інформаційних ресурсів.

REST має ряд керівних обмежень, які повинні бути виконання, щоб розроблений API можна було назвати RESTful. Частіше можна зустріти назву RESTful API. Розберемо окремо кожен з цих архітектурних обмежень, щоб дати розуміння того як має виглядати API в розроблюваній системі:

1. Клієнт-сервер. Це обмеження говорить про те, що необхідно дотримуватись розділення відповідальності між компонентами, мета яких зберігання та оновлення даних, і тими компонентами, що займаються відображенням даних на користувальницького інтерфейсі. Відокремлюючи задачі інтерфейсу користувача від проблем зберігання даних, ми покращуємо портативність цього самого інтерфейсу на багатьох платформах та покращуємо масштабованість.
2. Відсутність стану. Кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для розуміння цього самого запиту, Тобто запит не може покладатися на те, що сервер зберігає інформацію з попереднього запиту. Саме тому стан сесії має бути повністю збереженим на стороні клієнта.
3. Кешування. Обмеження кешу вимагають, щоб дані у відповіді на запит неявно або явно позначалися як “дозволяється для кешування” або як “не дозволяється для кешування”. Якщо відповідь є кешованою, то кеш клієнту надається право повторно використовувати ці дані відповіді для наступних, рівнозначних запитів і йому передається інформація про те, як довго ці дані можуть бути закешовані. Завдяки цьому з’являється можливість збільшувати продуктивність, уникаючи зайвих запитів на сервер. З іншого боку це зменшує надійність системи, адже кешовані дані можуть застаріти.
4. Однорідний інтерфейс. Застосовуючи цей принцип, спрощується загальна архітектура системи та покращується видимість взаємодій. Щоб отримати єдиний інтерфейс, для керування поведінкою компонентів потрібно кілька архітектурних обмежень. REST визначається чотирма обмеженнями інтерфейсу:
    - ідентифікація ресурсів – запит повинен точно визначати ресурс, що за ресурс очікується на отримання від серверу та якого формату повинна бути відповідь.
    - маніпуляція ресурсами через представлення – маючи представлення ресурсу, клієнт має достатньо інформації, щоб змінювати або видаляти цей ресурс.
    - самоописові повідомлення – інформації, що включає в себе повідомлення, має бути достатньо, щоб обробити повідомлення. Прикладом може служити назва

обробника повідомлення, який буде підходящим для формату отриманого ресурсу.

- гіпермедіа як рушій стану застосунку – клієнти змінюють стан системи тільки через дії, які динамічно визначені в гіпермедіа на сервері.

5. Шари абстракції. Відповідно до цього обмеження архітектура має поділятися на шари абстракцій, обмежуючи поведінку компонентів таким чином, щоб кожен компонент був обмежений безпосереднім шаром, з яким вони взаємодіють.
6. Запитування коду. Цей принцип дозволяє розширити функціональність клієнта за допомогою завантаження та виконання коду у вигляді аплетів чи скриптів. Це спрощує клієнт, зменшуючи кількість функцій, необхідних для попередньої реалізації. Проте цей принцип з часом став необов’язковим.

3.5 Висновки до розділу

Архітектура мікросервісу має важливе значення для гнучкої розробки та розгортання компонентів додатків, однак проектування мікросервісів для веб-додатків не є прямим завданням. Один з найкращих способів проектування мікросервісів - це розкласти монолітний прототип програми в мікросервіси.

Вирішено використати підхід до побудови архітектури “Monolith First”, з використанням якого розробка системи управління розповсюдженням та опрацюванням навчальної літератури почне свій розвиток доволі швидко і не потрібно витрачати додаткові ресурси на підтримку окремих сервісів.

В ході розробки буде створено моноліт з дотриманням модульності, який дозволить без зайвих зусиль виокремити модуль, що відповідає за збереження та опрацювання навчальної літератури.

РОЗДІЛ 4
РОЗРОБКА WEB-СИСТЕМИ

Для розробки даної системи було залучено велику кількість програмних засобів, практик та підходів. У цьому розділі розглянуто кожен з них докладно для розуміння необхідності його використання у даній web-системі.

4.1 Серверна частина системи

Почнемо з серверної частини додатку. Спираючись на архітектуру яка була обрана для побудови цієї інформаційної системи, було прийнято ряд рішень з приводу які технології мають бути використані для серверної частини застосунку. Окремі частини системи були винесені в окремі пункти для більш детального розгляду.

4.1.1 Мова програмування

Для будування web-системи з виконанням поставлених умов було обрано Java 11. Наступні пункти містять інформацію про бібліотеки, фреймворки та підходи, що дозволяють максимально ефективно побудувати таку систему та добре зарекомендували себе на практиці.

Оскільки мова є дуже популярною, у наявності в відкритому доступі є багато ресурсів, які можуть виступати гарним допоміжним заходом у проектуванні такої інформаційної системи. Багато технологій та підходів вже були багаторазово протестовані та використані на великих та довготривалих проектах. Java – це універсальна мова програмування. завдяки своїй міцності та масштабованості якої, застосунки на Java можна знайти на мобільних пристроях, комп’ютерах та на великих масштабних промислових серверах.

Однією з головних причин використання Java є її незалежність від платформи. Це означає, що застосунки написанні на Java, можна запускати на багатьох різних операційних системах. Після того як Java-код був написаний, його можна запустити на будь-якій платформі, проте для цього повинен бути встановлене середовище

виконання Java (JRE) на необхідній операційній системі, яку можна завантажити з офіційного веб-сайту Java.

Java по своїй суті є об'єктно-орієнтованою мовою, що дає розуміння про те, що програми на Java майже повністю складаються з об'єктів. У Java присутні наступні концепції об’єктно орієнтованого програмування, а саме:

- абстракція,
- інкапсуляція,
- успадкування,
- поліморфізм.

Такі ООП концепції Java дозволяють створювати робочі методи та змінні, а потім повторно безпечно використовувати частину з них або всі одразу.

Також у Java є велика екосистема, що постійно розвивається, створюючи нові бібліотеки, інструменти та фреймворки. На базі JVM також з'являються нові мови, такі як Scala, Groovy, тощо. Це також основна мова, яку Google використовує для розробки для Android, яка є найбільшою мобільною операційною системою.

4.1.2 Spring Framework

Коли створений клас має користуватися іншим класом-сервісом постає проблема у створені екземпляру цього сервісу. Мало того, що такий сервіс має бути створений в одному екземплярі так і створювати його в рамках класу не можна, адже порушується принципи єдиного обов'язку, підстановки Ліскова та інверсії залежностей. Для вирішення цієї проблеми необхідно використати контейнер інверсії контролю (IoC контейнер), такий контейнер містить у собі Spring Framework. Це одна з багатьох причин, чому необхідно використати Spring Framework.

Spring Framework пропонує комплексну модель програмування та конфігурації сучасних корпоративних застосунків написаних на Java. Ключовим елементом Spring є інфраструктурна підтримка на рівні програмного застосунку. Завдяки тому, що Spring фокусується на інфраструктурній частині додатків, розробники мають можливість зосередитись на бізнес-логіці уникаючи зайвих зв’язків із конкретними середовищами розгортання.

Завдяки вибраному підходу, зникає висока зв’язність на рівні класів. Як наслідок модульне тестування таких класів проходить набагато легше, адже завдяки слабкій залежності, з’являється можливість підміняти реалізації класів, таким чином ізолювати клас і тестувати тільки бізнес-логіку, що має відношення тільки до тестованого класу.

4.1.3 Аспектно-орієнтоване програмування

Spring Framework надає набагато більше можливостей ніж ін’єкція залежностей. Ще однією такою можливістю стало використання підходу аспектно-орієнтованого програмування (АОП). АОП називають парадигму програмування, особливістю якої є можливість впровадження наскрізної функціональності. Використовуючи АОП для модуляції окремих фрагментів логіки, з’являється можливість застосувати їх одразу до багатьох частин програми, при цьому не дублюючи код або створюючи жорсткі залежності.

Прикладами використання АОП в розробленій інформаційній системі стало логування, управління безпекою та рівнем доступу. При виконанні API запитів викликаються методи класів-контролерів, в методах яких є необхідність робити записи в журнал змін для фіксації результатів запитів. В явному вигляді прописувати логування не має потреби, адже усі результати API запитів будуть залоговані завдяки наскрізній функціональності, що реалізована в аспекті “LoggingAspect”. Для більшого розуміння концепцій АОП розберемо декілька окремо:

1. Точки з’єднання (joinpoints) – це чітко визначені точки в рамках додатку. Прикладом таких точок можуть служити виклик методу, ініціалізацію класів або інстанціювання об'єкта. Такі точки визначають місця в коді, в які можна додати необхідну логіку.

2. Порада (advice) – код, який виконується в певній точці приєднання. Поради, визначаються методами і можуть мати один з багатьох типів. порад, таких як:

- “before” – запустити перед виконанням методу,
- “after returning” – запустити після того, як метод поверне результат,
- “after throwing” – запустити після того, як метод кине виняток,

- “around” – виконати навколо методу, що поєднує всі три типи порад, описані вище.

3. Зріз (pointcuts) – це сукупність точок з’єднання. Зріз дає розуміння для системи, чи підходить точка з’єднання до поради.

4. Аспект – це поєднання порад та точкових зрізів, зібраних у класі. Ця комбінація призводить до визначення логіки, яка має бути додана під час виконання додатку і місця де вона повинна виконуватись.

Парадигма аспектно-орієнтованого програмування не намагається замінити парадигму об’єктно-орієнтоване програмування (ООП), навпаки, АОП доповнює її. ООП вирішує багато проблем та задач проте, об’єктно-орієнтованому підходу бракує, можливості впровадження наскрізної логіки.

4.1.4 Spring Data

Головне завдання Spring Data полягає у наданні чіткої та послідовної моделі для доступу до даних, при цьому не вимагаючи робити зміни до схеми бази даних. Це полегшує використання технологій доступу до баз даних різних типів.

Розглянемо які можливості надає Spring Data для розроблюваної системи [13]:

1. Потужний репозиторій та спеціальні абстракції для відображення таблиць в об’єкти.
2. Динамічне створення запитів, що базується на іменах методів репозиторіїв.
3. Реалізація базових класів доменів, для надання можливість використовувати базові властивості.
4. Підтримка прозорого аудиту сутностей,
5. Можливість інтеграції спеціального коду репозиторію
6. Spring інтеграція через Java-конфігурацію та XML-конфігурацію.
7. Розширена інтеграція з контролерами Spring MVC

Зазвичай шар DAO складається з безлічі кодових шаблонів, які можна спростити. Завдяки цьому зменшується кількість артефактів потрібно підтримувати та з’являється узгодженість моделей доступу до даних. Spring Data робить це спрощення і дає змогу повністю відійти від реалізації DAO вручну.

Існує специфікація під назвою “Java Persistence API” (JPA), що визначає стандарт управління реляційними даними в додатках для Java ORM фреймворків. Модель програмування Spring Data підтримує JPA специфікацію, для цього необхідно просто використати інтерфейс “JpaRepository”, що дасть можливість Spring Data знайти цей інтерфейс і автоматично створити для нього потрібну реалізацію.

4.1.5 Swagger

Згідно обраної архітектури для інформаційної системи необхідно створити RESTful API. Для цього необхідний інструмент, що дозволить розробляти, взаємодіяти та документувати API. Для таких цілей було прийнято рішення використати Swagger.

Swagger дозволяє описати структуру розроблюваного API. Завдяки здатність API описувати власну структуру, Swagger читаючи її, може автоматично створювати докладну та інтерактивну документацію цього самого API. Swagger вміє автоматично генерувати клієнтські бібліотеки для API на багатьох мовах. Swagger взаємодіє з API просячи його повернути YAML або JSON, що містить детальний опис всього API. Цей файл по суті є списком ресурсів API, який дотримується специфікації OpenAPI. Ця специфікація вимагає включити наступну інформацію [14]:

- Які всі операції підтримує API?
- Які параметри API і що він повертає?
- Чи потрібен для користування API якийсь дозвіл?
- Умови, контактна інформація та ліцензія на використання API.

Swagger надає можливість написати специфікацію для API вручну, проте для розробленої системи було вирішено створити її автоматично спираючись на примітки (анотації) розставлених у коді. На рисунку 4.1 зображено приклад згенерованої HTML сторінки, що містить всі розроблені API методи. На сформованій сторінці є можливість взаємодії з API через інтерфейс Swagger. Запити можна робити безпосередньо з користувальницького інтерфейсу та параметрів, які пропонуються для заповнення.

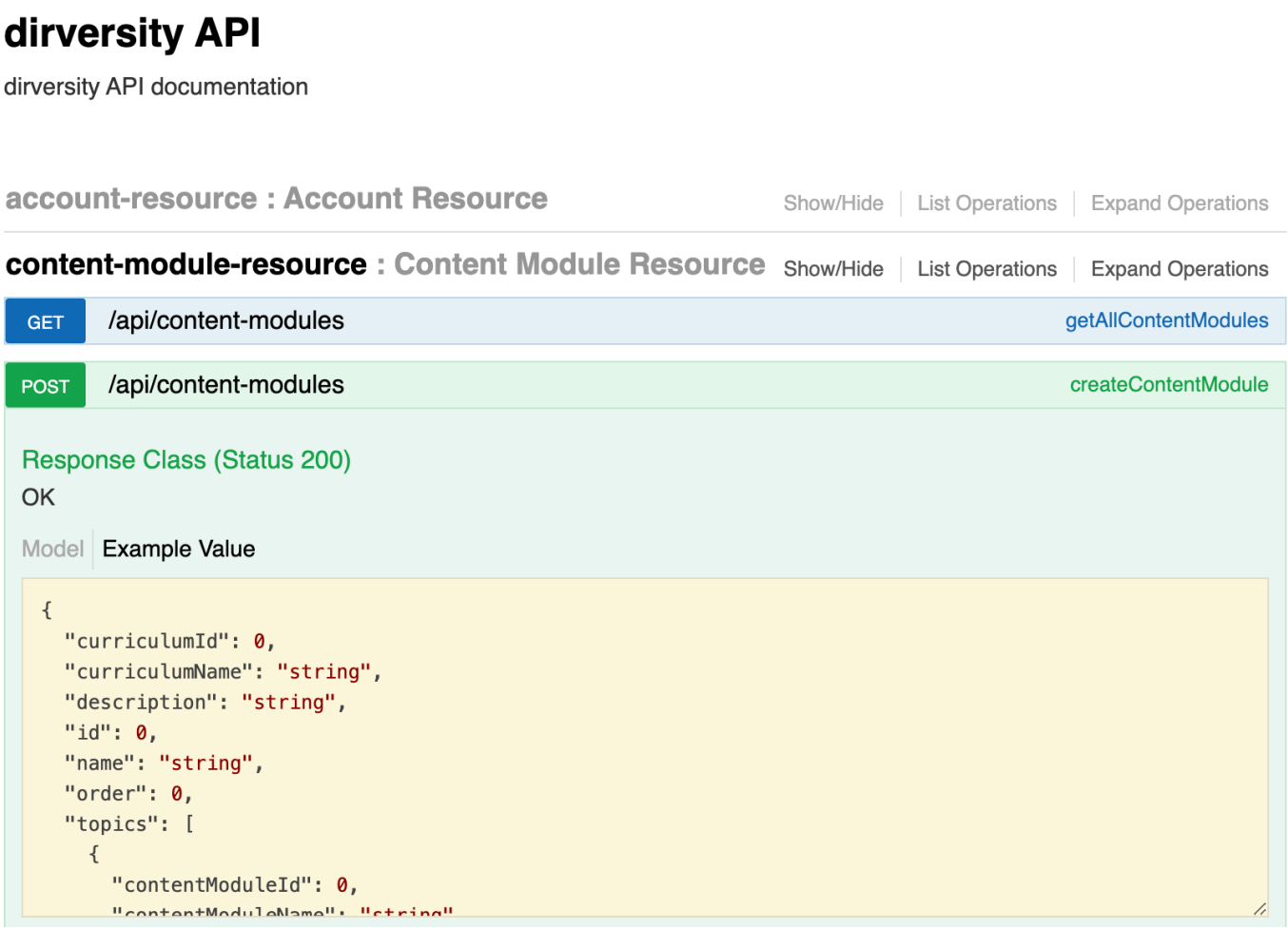

Рисунок 4.1 Приклад згенерованої API документації

4.1.6 Spring Task Scheduling

Одна з цілей розробленої інформаційної системи є автоматичне відправлення електронних листів з прикріпленими навчальними ресурсами. Враховуючи те, що в своїй основі серверна частина застосунку використовує Spring Framework, було використано одну з можливостей цього фреймворка, а саме “Планування завдань Spring” (Spring Task Scheduling).

Планування завдань в Spring складається з трьох частин:

1. визначення розкладу (тригера),

2. планувальника виконання завдання,

3. виконувана задача.

Для того, щоб відправляти листи автоматично у заданий користувачем час, в системі використовується заздалегідь сформована задача, мета якої перевіряти чи не

має бути в певному інтервалі відправлятися лист. Для цього система користуючись інструментами Spring, робить таку перевірку кожні 5 секунд.

Важливо те що, користувач також має можливість відправити листа натиснувши на кнопку “Відправити листа”. У цьому випадку клієнт зробить виклик на API, який в свою чергу відправити електронний лист.

4.2 Клієнтська частина системи

Для клієнтської частини було використано мову програмування TypeScript. Ця мова програмування має ряд переваг над JavaScript і в одночасно зберігає велику кількість бібліотек розроблених спеціально під JavaScript, адже TypeScript компілюється саме у JavaScript.

TypeScript забезпечує необов'язкову типізацію, класи та інтерфейси. Це дуже корисна перевага адже багато помилок під час роботи будуть одразу виявлені, ще на етапі компіляції.

Отже ось деякі з причин чому було обрано саме TypeScript:

- TypeScript підтримує бібліотеки JavaScript,
- JavaScript є підмножиною TypeScript,
- типізація присутня проте вона необов’язкова,
- код TypeScript може бути перетворений у звичайний JavaScript-код,
- надає можливість простіше структурувати код,
- краще підтримка об'єктно-орієнтованого підходу ніж в JavaScript

Для підвищення швидкості та якості розроблюваної системи було використано фреймворк Angular. Це платформа для веб-розробки, створена за допомогою мови програмування TypeScript, яка надає надійні інструменти для створення клієнтської сторони веб-додатків. Ось ряд переваг які надає Angular:

- Angular має дуже детальну та добре структуровану документацією, де розробники можуть знайти всю необхідну інформацію. Як результат, це полегшує процес розробки технічні рішення та вирішення виникаючих проблем.

- Підтримка компонентної архітектури. В рамках цієї архітектури додаток розділяється на незалежні логічні та функціональні компоненти. Як результат такі компоненти можна легко заміняти та роз'єднувати, чи повторно використовувати. Додатковою перевагою у наслідок незалежності компонентів стає полегшення тестування клієнтської частини системи.
- AOT-компіляція (Ahead-of-time compiler). AOT компілятор Angular перетворює TypeScript і HTML в JavaScript, що означає, що код компілюється до того, як браузер завантажить додаток. Як наслідок додаток буде відображатися набагато швидше.

4.3 Тестування web системи

Дуже важлива частина, адже підхід Monolith First сприяє до підготовки системи бути роздрібненою до мікросервісів. Одна з таких умов є добре покриття тестами. Важливо розуміти, що тестування не може служити доказом відсутності помилок у кодовій базі, проте воно допомагає відслідковувати якість написаного коду.

4.3.1 Модульне тестування серверної частини системи

Модульні тести займають найбільший прошарок серед усіх видів тестування web-системи. Основна ідея модульних тестів полягає в тому, щоб писати тести для кожної нетривіальною функції або методу. Це дозволяє досить швидко перевірити, чи не призвела чергові зміні внесені до кодової бази до регресії. Під регресією розуміється поява помилок в місцях програми, що вже були протестовані. Не менш важливу користь вони проявляють у процесі виявлення і усунення таких помилок.

Модульне тестування проходить за принципом "білого ящика", що визначає за мету тестування внутрішніх структур , а не її функціональність. Як наслідок модульні тести можуть послужити дуже хорошою документацією, адже у таких тестах чітко відображено де і з якими параметрами потрібно викликати методи протестованих класів та якої поведінки слід очікувати від виклику методів з тими чи іншими параметрами.

Одне з головних переваг від використання модульних тестів – це безпека при додаванні нової функціональності. Для написання таких тестів і отримання перелічених переваг було використано бібліотеку з відкритим кодом JUnit так як вона є найпоширенішим рішенням для написання модульних тестів на Java та має добре розписану документацію.

4.3.2 Інтеграційне тестування серверної частини системи

Інтеграційне тестування відіграє важливу роль у циклі розробки web-систем шляхом перевірки взаємодії окремих модулів системи. При інтеграційному тестуванні стоїть перш за все фокус на перевірку передачі даних між цими модулями.

Суть інтеграційного тестування, полягає в тому, щоб перевірити, чи окремо розроблені модулі працюють разом, як очікувалося. Виконується такі перевірки шляхом активації багатьох модулів та проведення тестів найвищого рівня, щоб переконатися, що вони працювали разом. [15]

В розробленій системі інтеграційні тести були розроблені за допомогою фреймворка Spring MVC Test.

4.3.3 Модульне тестування клієнтської частини системи

Клієнтська система також має покриття тестами. Модульні тести дозволять імітувати доступ до кінцевих точок REST програми. Це надає можливість протестувати рівень інтерфейсу користувача, не запускаючи серверну частину системи. Тести модулів інтерфейсу користувача (UI) використовують Jest.

Jest – це одна з найпопулярніших JavaScript бібліотек для створення, запуску та структурування тестів. Jest поширюється як пакет NPM, який можна встановити його в будь-якому JavaScript проекті, а тому і в розробленій web-системі, оскільки мова програмування TypeScript, що у ньому використовується компілюється у JavaScript.

Для того щоб запустити всі тести рівня інтерфейсу користувача необхідно просто запустити команду “npm test”.

4.3.4 Архітектурне тестування

Архітектура програмного забезпечення є важливою умовою для підтримки чистоти кодової бази, а також дотримання якості програмного забезпечення.

Архітектури web-системи переслідує цілі, виконуючи які з’являється можливість швидко додавати нові функції та виправляти помилки. Такі цілі мають наступний вигляд:

1. підтримувати ремонтопридатність системи,
2. здатність заміняти окремі частини системи,
3. здатність системи розширюватись.

На початку побудови системи взаємодія кожного з компонентів чітко визначена та структурована. Проте бувають випадки коли при зростані кодової бази системи починають порушуватись деякі з принципів описаних на початку побови проекту. Все більше і більше випадків, коли нові функції просто додаватимуться найшвидшим чином, що може бути погано з сторони архітектури. Як приклад може виникнути велика або неочевидна зв’язність і як наслідок деякі зміні можуть мати непередбачуваний вплив на будь-який інший компонент. Безпечний спосіб уникнути таких проблем – визначити компоненти системи в коді та правила для цих компонентів та іх взаємодії.

Для архітектурного тестування було використано бібліотеку ArchUnit. ArchUnit – це бібліотека для перевірки архітектури коду написаного на Java. Ця бібліотека може перевіряти залежності між шарами (layers), пакунками (packages) та класами, перевіряти наявність циклічних залежностей тощо. ArchUnit реалізує таку можливість завдяки аналізу байт-коду Java, імпорту всіх класів у структуру коду Java. Основна увага ArchUnit полягає в автоматичному тестуванні архітектури та правил кодування. [16]

В розробленій системі використовуються архітектурні тести з метою перевірки того, що жоден з Java-класів, що належать до сервісів або репозиторіїв не залежать від веб-рівню, до якого входять кінцеві REST точки (REST endpoints).

Такі тести мають дуже читабельний вигляд та зрозумілу структуру. На лістингу 4.2 зображено приклад тесту мета якого перевіряти відсутність у класах-сервісах залежності від веб-прошарку.

Лістинг 4.2 Приклад архітектурного тесту

```
@Test
void servicesShouldNotDependOnWebLayer() {
  JavaClasses importedClasses = new ClassFileImporter()
    .withImportOption(ImportOption.Predefined.DO_NOT_INCLUDE_TESTS)
    .importPackages("com.dirversity");

  noClasses()
    .that()
        .resideInAnyPackage("..service..")
    .should().dependOnClassesThat()
        .resideInAnyPackage("..com.dirversity.web..")
  .because("Services should not depend on web layer")
  .check(importedClasses);
}
```

4.4 Захист та безпека системи

Існує майже нескінченний перелік причин, чому безпека додатків важлива. І для того щоб розроблена система мала гарний захист, було використано ряд технік та підходів.

4.4.1 Управління користувачами

Для того щоб розділяти права доступу для різних груп користувачів було розроблено систему профілювання для користувачів. Розроблено наступні профілі:

- ROLE_ADMIN
- ROLE_CONTENT_MANAGER
- ROLE_SYSTEM_ADMIN
- ROLE_STUDENT
- ROLE_TEACHER

З допомогою профілювання можна з легкістю розділяти права доступу для кожного користувача, причому один користувач має можливість мати декілька профілів. Для кожного користувача можна змінювати профілі прямо у системі. Всі користувачі, що мають профілі ROLE_ADMIN або ROLE_SYSTEM_ADMIN мають доступ до панелі адміністратора.

У такій панелі міститься інструмент “Управління користувачами”, де можна надавати або позбавляти користувачів профілів. Також, за необхідності кожного

користувача можна деактивувати і доки його знову не активують, він не зможе проникнути в систему. Така функціональність надає можливість адміністратору обмежити доступ для користувача не видаляючи його системи, тим самим зберігаючи його данні. Кожне редагування або створення користувача буде зафіксовано шляхом зберігання даних про те хто і коли вніс зміни. Іншими словами проводиться аудит сутності користувача. На рисунку 4.3 зображено зовнішній вигляд інструмента “Управління користувачами”.

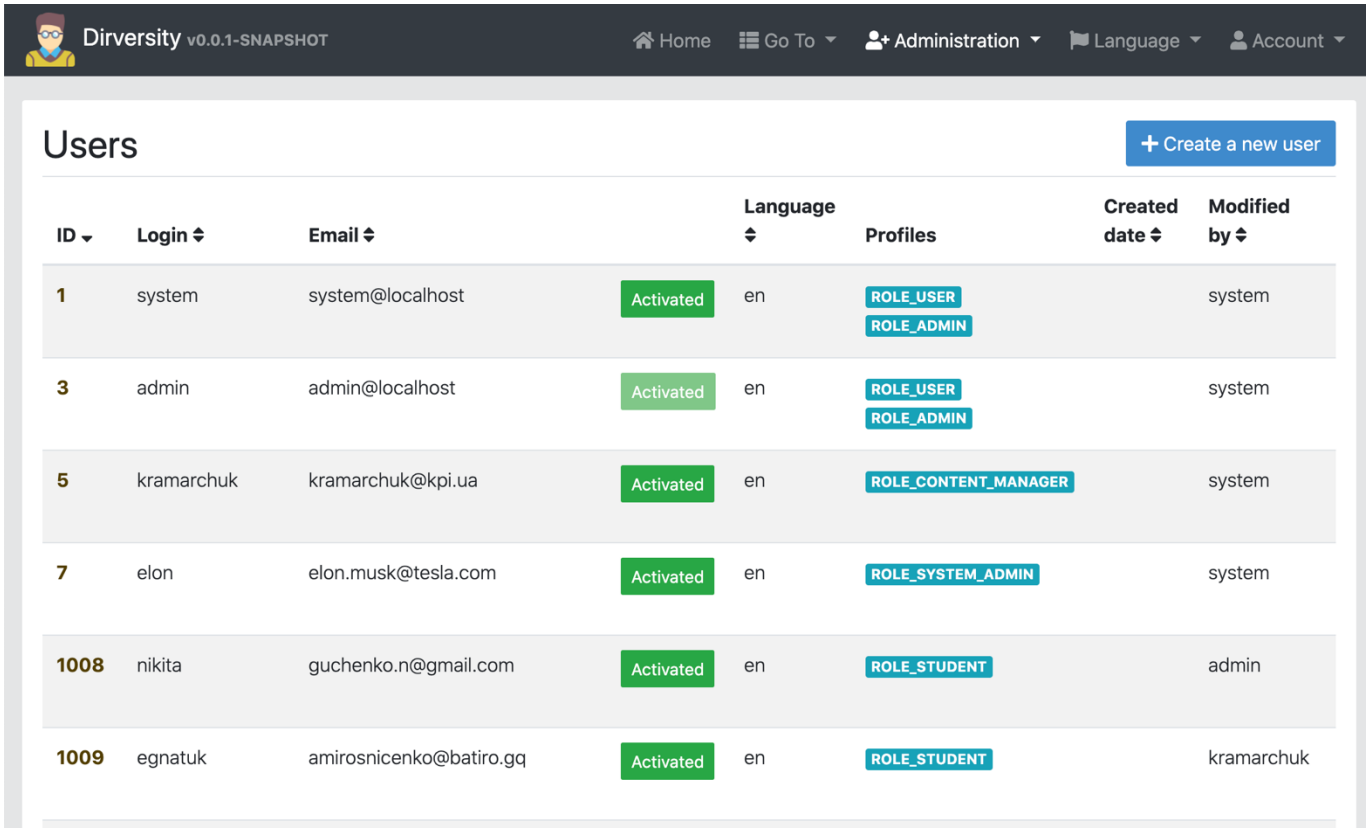

Рисунок 4.3 Інструмент “Управління користувачами”

До заповнення системи даними про користувачів, вже створено декілька користувачів, що потрібні для функціювання системи, а саме:

1. system – коли система робить автоматизовані задачі і проводить зміни до сутностей, що мають свій аудит, буде використовуватися дані саме цього користувача,
2. admin – необхідний для адміністрування системи,
3. anonymous user – дані цього користувача будуть використані для фіксації в аудиті змін сутностей анонімними користувачами.

4.4.2 JSON Web Tokens

JSON Web Token (JWT) автентифікація це захисний механізм, що не зберігає стан. У цьому рішенні використовується захищений токен, який містить ім’я та права користувача на вхід до системи. Коли токен назначений, користувач вже не може його змінити. На рисунку 4.4 зображено яким чином система автентифікує користувача.

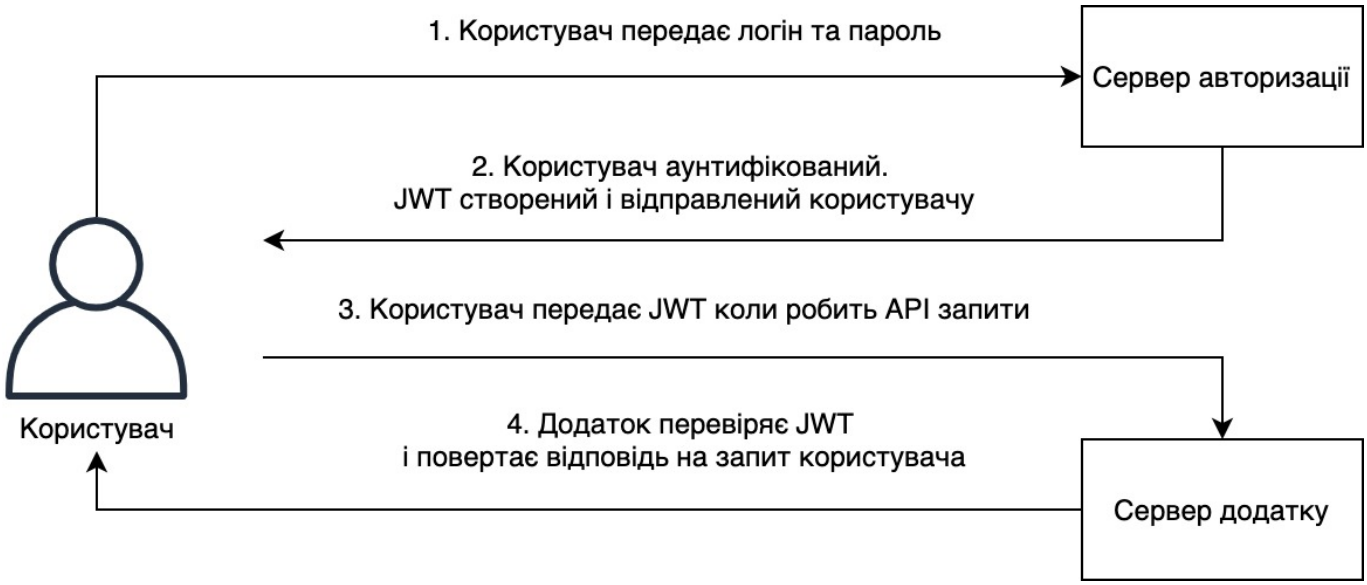

Рисунок 4.4 Схема отримання та використовування JWT

Система вміє по наданому їй токену отримувати дані про користувача. JWT складається з декількох частин.

1. Заголовок. Заголовок зазвичай складається з двох частин: типу токену (в цьому випадку – JWT), та використовуваного алгоритму підпису, такого як HMAC SHA256 або RSA.
2. Тіло. У тілі, зазвичай міститься інформація наступного виду:
    - ким виданий,
    - коли виданий,
    - строк дії токену (разом з інформацією про те коли виданий використовується для перевірки JWT),
    - Публічні мітки (ролі, ідентифікатор користувача і т. п.).

3. Підпис. Підпис необхідний для подальшої валідації JWT. Для створення підпису береться кодований заголовок, кодований тіло, секрет (secret) і використовуючи алгоритм, вказаний у заголовку, на виході маємо готовий підпис.

Система формує підпис наступним чином:

1. Система бере заголовок, кодований Base64url, та тіло, кодоване Base64url (заголовок кодований Base64url + "." + тіло кодоване Base64url), та хешує їх за допомогою вказаного у заголовку алгоритму SHA-256.
2. Система проводить шифрування за допомогою HMAC.
3. Base64url-кодує результат.

API декодує токен, в залежності від алгоритму і на підставу цієї інформації отримує інформацію про користувача. Кожен раз проходить валідація токену використовуючи дані з тіла JWT, а саме "строк дії" та "коли виданий". Коли час життя токену закінчується, додаток відповідає на запит з HTTP статус кодом 401, що означає, що необхідно отримати новий JWT.

Для того щоб перевірити підпис система робить наступні дії:

1. Перевіряє алгоритм створення підпису. З декодованого заголовка дістається властивість alg, що містить інформацію про використаний алгоритм. Система переконується, що це дозволений алгоритм і він використовується системою.
2. Система перевіряє підпис, щоб бути впевненою, що тіло не було змінено після підпису. Підпис створюється за допомогою за допомогою сегментів заголовка, тіла, секрету та алгоритму підписання, а це означає, що у системи є всі дані для того щоб створити новий підпис, кодований Base64url, за допомогою секрета (HS256) і переконатися, що він відповідає оригінальному підпису, включеному в отриманий JWT.

Для ілюстрації принципу декодування, на рисунку 4.5 JWT був розбитий на 3 частини і декодований за всіма правилами використовуючи алгоритм HS256.

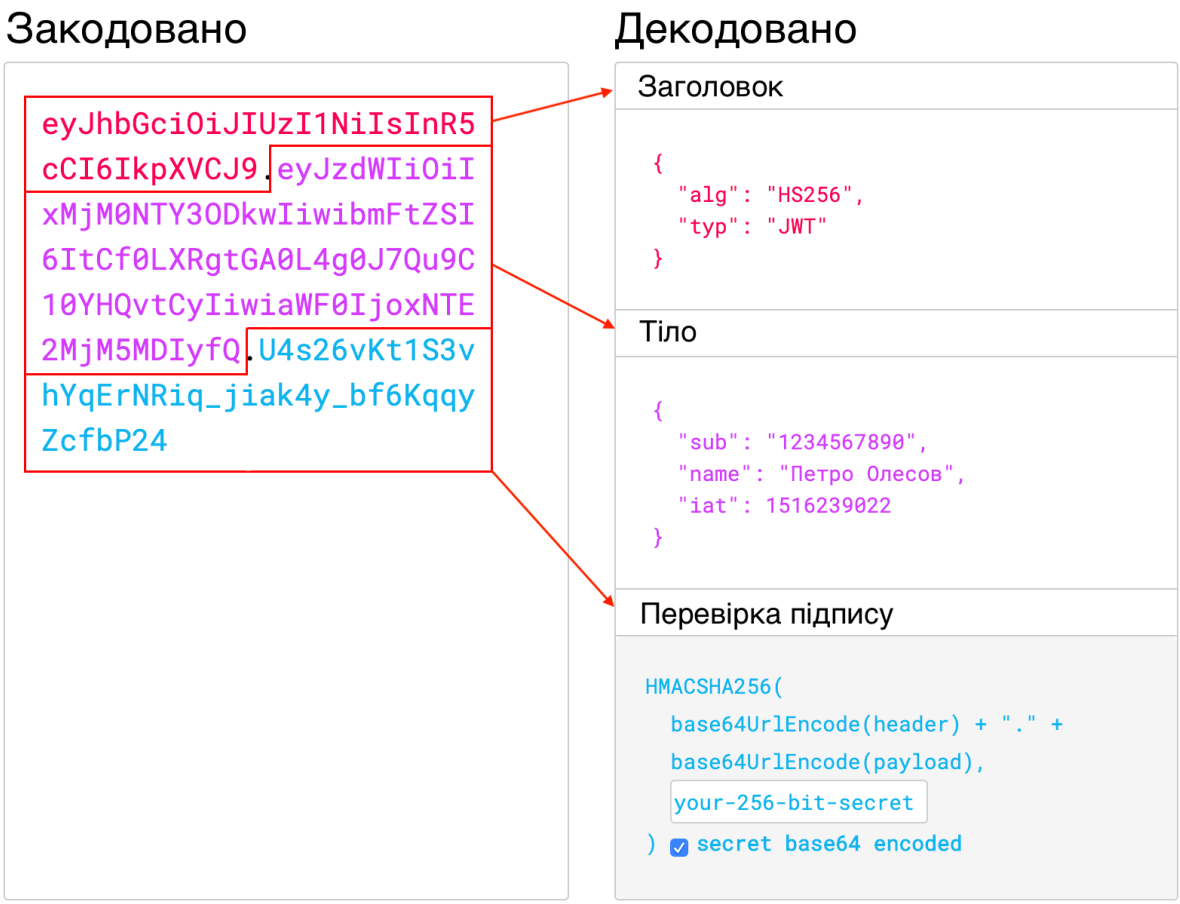

Рисунок 4.5 Декодування JWT

4.4.3 Spring Security

Spring Security – це потужний та налаштований фреймворк автентифікації та контролю доступу. Оскільки розроблена система використовує у своїй основі Spring то вибір саме Spring Security найбільш прийнятний. Як і всі проекти Spring, Spring Security має значну перевагу, а саме дуже просту можливість розширяти і додавати свої правила та умови.

У розроблюваній системі необхідна підтримка не тільки автентифікації але й авторизації. Розглянемо детальніше обидва процеси для визначення відмінностей.

Автентифікація стосується перевірки облікових даних, таких як ім'я користувача чи будь-якого вигляду ідентифікатор користувача та пароль для підтвердження особи. Незалежно від того, в публічній чи приватній мережі знаходиться користувач, система засвідчує особу користувача за допомогою паролів

для входу. Хоча зазвичай автентифікацію проводять за допомогою ідентифікатора користувача та пароля, існують й інші способи автентифікації.

У свою чергу авторизація відбувається після успішної автентифікації користувача, яка, таким чином, надає повний доступ до таких ресурсів інформаційної системи, однак авторизація підтверджує права на надання доступу до ресурсів лише після визначення того що користувач має відповідний права доступу. Тобто, авторизація – це процес визначення, чи має автентифікований користувач доступ до певних ресурсів.

Spring Security зосереджується не тільки на наданні автентифікації але і на авторизації для Java застосунків, тому саме його було обрано задля вирішення цих задач. Проте Spring Security має й інші можливості, такі як [17]:

4. Повна та розширена підтримка як для автентифікації, так і для авторизації.
5. Захист від атак, таких як фіксація сеансу, міжсайтова підробка запиту та інші.
6. Інтеграція API сервлетів.
7. Необов’язкова інтеграція з Spring Web MVC.

4.5 Зберігання даних

Для робленої інформаційної системи було використано декілька рішень для зберігання даних. Так як навчальні ресурси мають зберігатися окремо від інших даних було залучено різні сховища даних та розроблено окремі сервіси, які обробляють інформацію перед тим як вона потрапить до самого сховища.

4.5.1 Основна база даних

Перед цією базою даних постало завдання зберігати такі дані як:

1. дані про користувачів,
2. мета-даних про навчальні матеріали,
3. електронні листи,
4. права доступу користувачів,
5. робочі навчальні програми,
6. аудит,

7. групи користувачів,

8. типи ресурсних файлів.

Для вирішення проблеми зберігання даних такого виду було використано різні сховища даних для локальної розробки (locale database) та для фінального, доставленого рішення (production database).

Для готового, розгорнутого у кінцевому середовищі додатку було прийнято рішення використати систему управління реляційними базами даних MySQL. Проте це не означає, що та сама СУБД має бути використана і для локального середовища розробки. Саме тому для цього випадку було обрано H2, так як вона дуже швидка, а також її можна вбудовувати безпосередньо всередину Java-застосунку. Це необхідна перевага, адже в ході розробки набагато зручніше було користуватись саме режимом коли база даних створювалася в оперативній пам'яті під час запуску сервера застосунку.

Постало завдання керування змінами бази даних і синхронізувати їх з змінами у кодовій базі додатка. Оскільки у різних середовищах використовуються різні системи управління базами даних, було прийнято рішення використати бібліотеку, незалежної від бази даних з відкритим кодом, що розроблена для відстеження, застосування та управління змінами схеми бази даних під назвою Liquibase.

Liquibase – це рішення управління змінами схеми баз даних, що дозволяє легко керувати змінами бази даних і як наслідок її версійністю. Ця бібліотека надає наступні можливості:

1. Дає можливість розгорнути зміни бази даних та додатків разом. Таким чином всі зміни будуть завжди залишалися синхронізованими.

2. Автоматично генерує SQL запити,

3. Дозволяє використовувати підлаштовувати виконування скриптів залежно від контексту.

4. Створює контроль версій над зміни схеми бази даних.

5. Дозволяє легко проводити відкат змін.

6. Допомагає швидше усувати помилки пов’язаних з базами даних.

Для розробленої web-системи усі зміни в схемі бази даних зберігаються в текстових XML файлах. В таких файлах містяться сутність “databaseChangeLog” яка в собі містить набір сутностей “changeSet”, які містять в собі набір маніпуляцій над схемою бази даних. Кожен changeSet ідентифікуються за допомогою тегів “id” та “author”. У кожній базі даних Liquibase автоматично створює таблицю “DatabaseChangeLog” та “DatabaseChangeLogLock” де зберігається перелік усіх застосованих змін для того щоб визначити, які нові зміни мають бути застосовані. Такий підхід дозволяє тримати та відстежувати всю історію змін зроблених над базою даних.

Оскільки база даних H2 створюється в оперативній пам’яті, постала проблема відсутності тестових даних, так як всі вони видалялись кожен раз коли сервер додатку припиняв свою роботу. Для вирішення цієї проблеми було використано функціональність Liquebase, а саме додавання до списку змін XML-тег “loadData”, у якому прописано з якого файлу треба брати інформацію, у яку таблицю. Треба її зберігати та в рамках якого контексту треба виконувати завантаження даних. Тобто якщо відповідний контекст відсутній, то і завантаження даних відбуватися не буде. Завдяки цьому можна налаштовувати в якому середовищі потрібно саме ці, тестові дані додавати до бази даних а в якому ні. Контекст задається у відповідному до середовища yml-файлі.

4.5.2 Сховище даних для навчальних матеріалів

Завдяки побудованій архітектурі сховище для зберігання навчальних матеріалів може мати будь який вигляд. Головна умова полягає у тому, що сервіс який працює з цією базою даних має реалізовувати відповідний API

Для навчальних матеріалів було обрано Google Drive. Таке рішення має декілька переваг такий як гарно структурований Google Drive API, яке дозволяє створювати додатки, які використовують хмарне сховище Google Диска. За його допомогою розроблювана система може швидко провести інтеграцію з Google Диском.

Google Drive API – це REST API, який дозволяє використовувати сховище Google Drive із свого додатка. За допомогою цього API можна:

- Завантажувати файли з Google Drive і завантажте файли на Google Drive.
- Проводити пошук по файлам та папкам, які зберігаються на Google Drive.
- Створювати складні пошукові запити, які повертають будь-яке з полів метаданих файлів.
- Надавати сумісний доступ до даних.
- Чітко визначати у кого які є права на читання або редагування ресурсів.
- Генерувати посилання на ресурси які можуть використовуватись розробленою системою для поширення навчальних матеріалів.
- Користуватися імпортованими навчальними даними через інтерфейс Google Drive, що надає багато переваг, таких як редагування даних напряму у хмарному сховища, без попереднього завантаження.

Для того щоб завантажити дані до Google Drive, система може обирати серед трьох можливих режимів завантаження, а саме:

- Просте завантаження. Для цього під час запиту до Google Drive необхідно вказати параметр “uploadType” як “media”. Цей тип завантаження використовується для швидкого перенесення невеликого медіа-файлу без надання метаданих. Це файли розміром менше 5 Мб.
- Завантаження ресурсів за допомогою подрібнення. Для цього система виставляє параметр “uploadType” як “multipart”. Такий тип використовується для швидкого перенесення невеликого файлу та його метаданих, що описують файл, в одному запиті. Розмір файлу має бути 5 Мб або менше.
- Завантаження з можливістю відновлення. Щоб використати саме такий ти завантаження необхідно вказати параметр “uploadType” як “resumable”. Необхідний такий тип завантаження для великих файлів, а саме більше ніж 5 Мб, особливо коли є велика ймовірність перерви в мережі. Завантаження з можливістю відновлення є хорошим вибором для більшості програм,

оскільки вони також працюють для невеликих файлів з мінімальними витратами на один додатковий запит HTTP за завантаження.

4.6 Розгортання системи

З-поміж можливих платформ для розгортання було обрано хмарну PaaS-платформу (Platform as a Service) Heroku. Хмарні обчислення – це парадигма інформаційних технологій, а саме модель для надання повного доступу до спільних пулів конфігуруваних ресурсів. До таких ресурсів, можна віднести комп'ютерні мережі, сервери, сховища, програми та послуги. Хмарні платформи, такі як Heroku можуть бути швидко надані з мінімальними зусиллями на управління.

Ось деякі з переваг, що надає Heroku:

1. Дозволяє розробнику зосереджуватися на коді замість інфраструктури.
2. Наявний моніторинг додатків.
3. Надає просту горизонтальну та вертикальну масштабованість.
4. Надає провідні інструменти платформи.
5. Допомагає зосередитись на інноваціях, а не на операціях.
6. Надає можливість швидко керувати життєвим циклом програми.
7. Пропонує потужну інформаційну панель та CLI.
8. Підтримує життєдіяльність розгорнутої системи задля збереження показників у нормальному стані.
9. Надає набір автоматизованих функцій, включаючи масштабування, конфігурацію, налаштування тощо.

Для полегшення процесу розгортання використано платформу розробки JHipster, адже він має підгенератор, що дозволяє швидко розгорнути програму в Heroku Cloud. Для того щоб не зберігати "чутливу" інформацію в кодовій базі було використано змінні середовища.

Після розгортання інформаційна система доступна по посиланню: https://dirversity-demo.herokuapp.com/.

4.7 Висновки до розділу

В цьому розділі було проведено огляд програмних засобів, практик та підходів, що були залучені задля розробки всіх функцій системи. Кожен з них докладно розглянуто задля розуміння необхідності його використання у розробленій інформаційній системі.

Кожен з програмних засобів чи підходів було залучено з урахуванням визначеної архітектури. Враховуючи те, що згідно архітектури в системі чітко відділена серверна частина від клієнтської, було розглянуто обидві частини окремо.

Не менш важливою темою є захист web-системи. Для того щоб розроблена система мала гарний захист, було використано ряд підходів та програмних засобів, які були детально розглянуті у цьому розділі.

Тестування є невід'ємною складовою в процесі розробки програмного забезпечення. Тим більше, одна з таких умов визначеної архітектурі є покриття тестами, адже воно допомагає відслідковувати якість написаного коду. Було тестування різних видів, з них:

1. модульне тестування серверної частини системи,
2. модульне тестування клієнтської частини системи,
3. інтеграційне тестування серверної частини системи,
4. архітектурне тестування.

Для інформаційної системи використовується одразу декілька рішень для зберігання даних. Оскільки навчальні ресурси мають зберігатися окремо від інших даних було залучено різні сховища даних та розроблено окремі сервіси, що обробляють інформацію перед тим як вона потрапить до самого сховища. Кожне з них детально розглянуто в розділі.

Вже розроблену систему було розгорнуто на хмарній платформі Heroku.

ВИСНОВКИ

В ході виконання роботи було створено систему, що автоматизує процес поширення навчальної літератури, робить його простим у використані та легким у впроваджені. Було проаналізовано окрім конкретних інформаційних систем, цілі категорії інформаційних систем з ціллю виявити їх сильні та слабкі сторони, що було враховано при проектуванні системи.

Розроблена система знімає з викладача відповідальність в управлінні розсилкою навчальних матеріалів. Система бере на себе задачу в потрібний день відправляти студентам книги, конспекти, лабораторні роботи та інші ресурси. Система підвищує якість та швидкість опанування студентами навчальних матеріалів, у порівнянні з традиційними підходами.

Завдяки єдиній структурі електронних листів, що використовується системою, кожен студент може відфільтрувати листи так як він забажає і знайти необхідні йому матеріали. Це вкрай важливо тому що легкий доступ до навчальної інформації має один з найвищих пріоритетів.

Також було враховано можливість використання навчальним закладом нестандартних рішень для зберігання даних. Це стало одним із вирішальних аргументів у виборі підходу до побудови архітектури інформаційної системи.

Освоєно новий підхід до розробки бази даних, а саме версійний контроль зміни схеми бази даних. Це спростило процес побудови бази даних на всьому етапі розробки web-системи.

Розроблена інформаційна система легко впроваджується за рахунок того, що студенту необов’язково мати обліковий запис в системі, достатньо мати пошту, на яку буде приходити вся необхідна інформація з контентом частково генерованим системою і частково викладачем.

Виконано усі задачі, необхідні для досягнення мети роботи, а саме web-система, що автоматизує процес поширення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А. М. Береза Основи створення інформаційних систем: Навч. посібник. 2 видання,перероблене і доповнене – К.: КНЕУ, 2001. – 204 с.

2. Pardo-Bunte M. Google Classroom Review [Електронний ресурс] / Melissa Pardo-Bunte // Better Buys. – 2020. – Режим доступу до ресурсу: https://www.betterbuys.com/lms/reviews/google-classroom/.

3. Google Classroom Review: Pros And Cons Of Using Google Classroom In eLearning [Електронний ресурс] // eLearning Industry. – 2015. – Режим доступу до ресурсу: https://elearningindustry.com/google-classroom-review-pros-and-cons-of-using-google-classroom-in-elearning.

4. Pardo-Bunte M. Moodle Review [Електронний ресурс] / Melissa Pardo-Bunte // Better Buys. – 2020. – Режим доступу до ресурсу: https://www.betterbuys.com/lms/reviews/moodle/.

5. Rossi G. Web Engineering: Modelling and Implementing Web Applications / G. Rossi, P. Oscar, S. Daniel. – New York: Springer-Verlag, 2008. – 476 с.

6. Carnell J. Spring Microservices in Action / John Carnell., 2017. – 384 с.

7. Fowler M. Microservices. a definition of this new architectural term [Електронний ресурс] / Martin Fowler. – 2014. – Режим доступу до ресурсу: https://martinfowler.com/articles/microservices.html.

8. Fowler M. Monolith First [Електронний ресурс] / Martin Fowler. – 2015. – Режим доступу до ресурсу: https://martinfowler.com/bliki/MonolithFirst.html#footnote-typical-monolith.

9. Web Services Architecture [Електронний ресурс] / [D. Booth, H. Haas, F. McCabe та ін.] // World Wide Web Consortium. – 2004. – Режим доступу до ресурсу: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211.

10. Pearlman S. What are APIs and how do APIs work? [Електронний ресурс] / Shana Pearlman // MuleSoft Blog. – 2016. – Режим доступу до ресурсу: https://blogs.mulesoft.com/biz/tech-ramblings-biz/what-are-apis-how-do-apis-work/.

11. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship / Robert Martin., 2008. – 464 с.

12. Liskov B. Behavioral Subtyping Using Invariants and Constraints / B. Liskov, J. Jeannette., 1999. – 22 с.

13. Spring Data [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://spring.io/projects/spring-data.

14. What Is Swagger? [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://swagger.io/docs/specification/2-0/what-is-swagger/.

15. Fowler M. Integration Test [Електронний ресурс] / Martin Fowler. – 2018. – Режим доступу до ресурсу: https://martinfowler.com/bliki/IntegrationTest.html.

16. Gafert P. ArchUnit User Guide [Електронний ресурс] / Peter Gafert – Режим доступу до ресурсу: https://www.archunit.org/userguide/html/000_Index.html.

17. Spring Security [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://spring.io/projects/spring-security.

ДОДАТОК А

Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

Специфікація

“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б

Аркушів 2

Київ – 2020 року

Позначення	Найменування	Примітки
Документація		
“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 12-1	Resource.java	Основний компонент, що містить всю інформацію про ресурси
“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 12-2	ResourceRepository.java	Компонент інтеграції з базою даних
“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 12-3	ResourceServiceImpl.java	Компонент, що містить функції з управління ресурсами
“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 12-4	ResourceResource.java	Компонент, що оброблює REST запити

ДОДАТОК Б

Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

Лістинг програми

“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 12-1

Аркушів 11

Київ – 2020 року

Resource.java

```
package com.dirversity.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

/**
 * A Resource.
 */
@Entity
@Table(name = "resource")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public class Resource extends AbstractAuditingEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "author")
    private String author;

    @Column(name = "access_url")
    private String accessUrl;

    @Column(name = "file_id")
    private String fileId;

    @ManyToMany
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
```

```
    @JoinTable(name = "resource_resource_type",
            joinColumns = @JoinColumn(name = "resource_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "resource_type_id", referencedColumnName =
"id"))
    private Set<ResourceType> resourceTypes = new HashSet<>();

    @ManyToMany
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    @JoinTable(name = "resource_rules",
            joinColumns = @JoinColumn(name = "resource_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "rules_id", referencedColumnName = "id"))
    private Set<Rule> rules = new HashSet<>();

    @ManyToMany(fetch = FetchType.EAGER)
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    @JoinTable(name = "resource_topic",
            joinColumns = @JoinColumn(name = "resource_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "topic_id", referencedColumnName = "id"))
    private Set<Topic> topics = new HashSet<>();

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public Resource name(String name) {
        this.name = name;
        return this;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAuthor() {
        return author;
    }

    public Resource author(String author) {
        this.author = author;
        return this;
    }

    public void setAuthor(String author) {
```

```
    this.author = author;
}

public String getAccessUrl() {
    return accessUrl;
}

public Resource accessUrl(String accessUrl) {
    this.accessUrl = accessUrl;
    return this;
}

public void setAccessUrl(String accessUrl) {
    this.accessUrl = accessUrl;
}

public String getFileId() {
    return fileId;
}

public Resource fileId(String fileId) {
    this.fileId = fileId;
    return this;
}

public void setFileId(String fileId) {
    this.fileId = fileId;
}

public Set<ResourceType> getResourceTypes() {
    return resourceTypes;
}

public Resource resourceTypes(Set<ResourceType> resourceTypes) {
    this.resourceTypes = resourceTypes;
    return this;
}

public Resource addResourceType(ResourceType resourceType) {
    this.resourceTypes.add(resourceType);
    resourceType.getResources().add(this);
    return this;
}

public Resource removeResourceType(ResourceType resourceType) {
    this.resourceTypes.remove(resourceType);
    resourceType.getResources().remove(this);
    return this;
}

public void setResourceTypes(Set<ResourceType> resourceTypes) {
```

```
        this.resourceTypes = resourceTypes;
    }

    public Set<Rule> getRules() {
        return rules;
    }

    public Resource rules(Set<Rule> rules) {
        this.rules = rules;
        return this;
    }

    public void setTopics(Set<Topic> topics) {
        this.topics = topics;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof Resource)) {
            return false
        }
        return id != null && id.equals(((Resource) o).id);
    }

    @Override
    public int hashCode() {
        return 31;
    }
}
```

ResourceRepository.java

```
package com.dirversity.repository;
import com.dirversity.domain.Resource;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

/**
 * Spring Data  repository for the Resource entity.
 */
```

```
@Repository
public interface ResourceRepository extends JpaRepository<Resource, Long> {

    @Query(value = "select resource from Resource resource where resource.createdBy =
?#{principal.username}",
        countQuery = "select count(distinct resource) from Resource resource")
    Page<Resource> findByPublisherIsCurrentUser(Pageable pageable);

    @Query(value = "select distinct resource from Resource resource left join fetch
resource.resourceTypes left join fetch resource.rules left join fetch resource.topics",
        countQuery = "select count(distinct resource) from Resource resource")
    Page<Resource> findAllWithEagerRelationships(Pageable pageable);

    @Query("select distinct resource from Resource resource left join fetch resource.resourceTypes left
join fetch resource.rules left join fetch resource.topics")
    List<Resource> findAllWithEagerRelationships();

    @Query("select resource from Resource resource left join fetch resource.resourceTypes left join fetch
resource.rules left join fetch resource.topics where resource.id =:id")
    Optional<Resource> findOneWithEagerRelationships(@Param("id") Long id);

}
```

ResourceServiceImpl.java

```
package com.dirversity.service.impl;

import com.dirversity.service.ResourceService;
import com.dirversity.domain.Resource;
import com.dirversity.repository.ResourceRepository;
import com.dirversity.service.dto.ResourceDTO;
import com.dirversity.service.mapper.ResourceMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

/**
 * Service Implementation for managing {@link Resource}.
 */
@Service
@Transactional
public class ResourceServiceImpl implements ResourceService {
```

```
    private final Logger log = LoggerFactory.getLogger(ResourceServiceImpl.class);

    private final ResourceRepository resourceRepository;

    private final ResourceMapper resourceMapper;

    public ResourceServiceImpl(ResourceRepository resourceRepository, ResourceMapper
resourceMapper) {
        this.resourceRepository = resourceRepository;
        this.resourceMapper = resourceMapper;
    }

    /**
     * Save a resource.
     *
     * @param resourceDTO the entity to save.
     * @return the persisted entity.
     */
    @Override
    public ResourceDTO save(ResourceDTO resourceDTO) {
        log.debug("Request to save Resource : {}", resourceDTO);
        Resource resource = resourceMapper.toEntity(resourceDTO);
        resource = resourceRepository.save(resource);
        return resourceMapper.toDto(resource);
    }

    /**
     * Get all the resources.
     *
     * @param pageable the pagination information.
     * @return the list of entities.
     */
    @Override
    @Transactional(readOnly = true)
    public Page<ResourceDTO> findAll(Pageable pageable) {
        log.debug("Request to get all Resources");
        return resourceRepository.findAll(pageable)
            .map(resourceMapper::toDto);
    }

    /**
     * Get all the resources with eager load of many-to-many relationships.
     *
     * @return the list of entities.
     */
    public Page<ResourceDTO> findAllWithEagerRelationships(Pageable pageable) {
        return resourceRepository.findAllWithEagerRelationships(pageable).map(resourceMapper::toDto);
    }

    @Override
    public Page<ResourceDTO> findAllResourcesCreatedByCurrentUser(Pageable pageable) {
```

```
        log.debug("Request to get all Resources for the user");
        return resourceRepository.findByPublisherIsCurrentUser(pageable)
            .map(resourceMapper::toDto);
    }


    /**
     * Get one resource by id.
     *
     * @param id the id of the entity.
     * @return the entity.
     */
    @Override
    @Transactional(readOnly = true)
    public Optional<ResourceDTO> findOne(Long id) {
        log.debug("Request to get Resource : {}", id);
        return resourceRepository.findOneWithEagerRelationships(id)
            .map(resourceMapper::toDto);
    }

    /**
     * Delete the resource by id.
     *
     * @param id the id of the entity.
     */
    @Override
    public void delete(Long id) {
        log.debug("Request to delete Resource : {}", id);
        resourceRepository.deleteById(id);
    }
}
```

ResourceResource.java

```
package com.dirversity.web.rest;

import com.dirversity.security.AuthoritiesConstants;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import java.util.List;
import java.util.Optional;

/**
 * REST controller for managing {@link com.dirversity.domain.Resource}.
 */
```

```
@RestController
@RequestMapping("/api")
public class ResourceResource {

    private final Logger log = LoggerFactory.getLogger(ResourceResource.class);

    private static final String ENTITY_NAME = "resource";

    @Value("${jhipster.clientApp.name}")
    private String applicationName;

    private final ResourceService resourceService;
    private final UserService userService;
    private final CloudStorageService cloudStorageService;

    public ResourceResource(ResourceService resourceService, UserService userService,
CloudStorageService cloudStorageService) {
        this.resourceService = resourceService;
        this.userService = userService;
        this.cloudStorageService = cloudStorageService;
    }

    /**
     * {@code POST  /resources} : Create a new resource.
     *
     * @param resourceDTO the resourceDTO to create.
     * @return the {@link ResponseEntity} with status {@code 201 (Created)} and with body the new
resourceDTO, or with status {@code 400 (Bad Request)} if the resource has already an ID.
     * @throws URISyntaxException if the Location URI syntax is incorrect.
     */
    @PostMapping("/resources")
    public ResponseEntity<ResourceDTO> createResource(@RequestBody ResourceDTO resourceDTO)
throws URISyntaxException {
        log.debug("REST request to save Resource : {}", resourceDTO);
        if (resourceDTO.getId() != null) {
            throw new BadRequestAlertException("A new resource cannot already have an ID",
ENTITY_NAME, "idexists");
        }

//        ToDo Refactor this to make it less coupled
        uploadResourceToCloudStorage(resourceDTO);

        ResourceDTO result = resourceService.save(resourceDTO);
        return ResponseEntity.created(new URI("/api/resources/" + result.getId()))
            .headers(HeaderUtil.createEntityCreationAlert(applicationName, true, ENTITY_NAME,
result.getId().toString()))
            .body(result);
    }

    private Instant getNow() {
        LocalDateTime dateTime = LocalDateTime.now();
```

```
        return dateTime.atZone(ZoneId.systemDefault()).toInstant();
    }

    /**
     * {@code PUT  /resources} : Updates an existing resource.
     *
     * @param resourceDTO the resourceDTO to update.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the updated
resourceDTO,
     * or with status {@code 400 (Bad Request)} if the resourceDTO is not valid,
     * or with status {@code 500 (Internal Server Error)} if the resourceDTO couldn't be updated.
     * @throws URISyntaxException if the Location URI syntax is incorrect.
     */
    @PutMapping("/resources")
    public ResponseEntity<ResourceDTO> updateResource(@RequestBody ResourceDTO resourceDTO)
throws URISyntaxException {
        log.debug("REST request to update Resource : {}", resourceDTO);
        if (resourceDTO.getId() == null) {
            throw new BadRequestAlertException("Invalid id", ENTITY_NAME, "idnull");
        }
        uploadResourceToCloudStorage(resourceDTO);
        ResourceDTO result = resourceService.save(resourceDTO);
        return ResponseEntity.ok()
            .headers(HeaderUtil.createEntityUpdateAlert(applicationName, true, ENTITY_NAME,
resourceDTO.getId().toString()))
            .body(result);
    }

    private void uploadResourceToCloudStorage(ResourceDTO resourceDTO) {
        if (resourceDTO.getData() != null && resourceDTO.getDataContentType() != null) {
            cloudStorageService
                .uploadFileData(resourceDTO.getDataContentType(), resourceDTO.getDataDisplayName(),
resourceDTO.getData())
                .ifPresent(file -> {
                    resourceDTO.setFileId(file.getId());
                    resourceDTO.setAccessUrl(file.getWebViewLink());
                });
        }
    }


    /**
     * {@code GET  /resources} : get all the resources.
     *
     * @param pageable  the pagination information.
     * @param eagerload flag to eager load entities from relationships (This is applicable for many-to-
many).
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and the list of resources in
body.
     */
    @GetMapping("/resources")
```

```
    public ResponseEntity<List<ResourceDTO>> getAllResources(Pageable pageable,
@RequestParam(required = false, defaultValue = "false") boolean eagerload) {
        log.debug("REST request to get a page of Resources");

        Page<ResourceDTO> page;
        if (SecurityUtils.isCurrentUserInRole(AuthoritiesConstants.ADMIN)) {
            if (eagerload) {
                page = resourceService.findAllWithEagerRelationships(pageable);
            } else {
                page = resourceService.findAll(pageable);
            }
        } else {
            page = resourceService.findAllResourcesCreatedByCurrentUser(pageable);
        }


        HttpHeaders headers =
PaginationUtil.generatePaginationHttpHeaders(ServletUriComponentsBuilder.fromCurrentRequest(),
page);
        return ResponseEntity.ok().headers(headers).body(page.getContent());
    }

    /**
     * {@code GET  /resources/:id} : get the "id" resource.
     *
     * @param id the id of the resourceDTO to retrieve.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the
resourceDTO, or with status {@code 404 (Not Found)}.
     */
    @GetMapping("/resources/{id}")
    public ResponseEntity<ResourceDTO> getResource(@PathVariable Long id) {
        log.debug("REST request to get Resource : {}", id);
        Optional<ResourceDTO> resourceDTO = resourceService.findOne(id);
        return ResponseUtil.wrapOrNotFound(resourceDTO);
    }

    /**
     * {@code DELETE  /resources/:id} : delete the "id" resource.
     *
     * @param id the id of the resourceDTO to delete.
     * @return the {@link ResponseEntity} with status {@code 204 (NO  CONTENT)}.
     */
    @DeleteMapping("/resources/{id}")
    public ResponseEntity<Void> deleteResource(@PathVariable Long id) {
        log.debug("REST request to delete Resource : {}", id);
        resourceService.delete(id);
        return ResponseEntity.noContent().headers(HeaderUtil.createEntityDeletionAlert(applicationName,
true, ENTITY_NAME, id.toString())).build();
    }
}
```

ДОДАТОК В

Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

Опис програмного коду

“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 13-1

Аркушів 10

Київ – 2020 року

АНОТАЦІЯ

Додаток містить опис web-системи з централізованого управління розповсюдженням та опрацюванням навчальної літератури, що виконує деякі із завдань, поставлених в розділі 1, а саме:

— Завантаження навчальних матеріалів;

— Автоматичне відправлення навчальних матеріалів;

— Асоціація навчальних матеріалів з темами робочої навчальної програми.

Додаток розроблений за допомогою мови програмування Java у середовищі програмування IntelliJ IDEA.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис основного компоненту web-системи з централізованого управління розповсюдженням та опрацюванням навчальної літератури, що виконує деякі із завдань, поставлених в розділі 1. У додатку Б міститься програмний код компоненту.

Система для роботи потребує браузер, що підтримує JavaScript та Cookies.

Додаток розроблений за допомогою мови програмування Java у середовищі програмування IntelliJ IDEA.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Система надає функціонал для роботи з навчальними матеріалами, а саме:

— завантаження навчальних матеріалів у хмарне сховище;

— отримання посилання на завантажений навчальний матеріал;

— асоціація навчальних матеріалів з темами робочої навчальної програми.;

— поширення навчальних матеріалів у задану дату та час.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Згідно з архітектурою, додаток складається з трьох головних модулів:

— шар сервлетів;

— шар моделей;

— шар зв’язку з базою даних.

Шар сервлетів містить в собі декілька класів, що слугують для виклику зовнішніх методів із сторінок.

Шар моделей містить в собі класи та інтерфейси, що використовуються для моделювання сутностей, якими оперує додаток.

Шар зв’язку з базою даних містить класи з методами для створення, видалення та оновлення полів у базі.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання додатку користувач повинен мати персональний комп’ютер або мобільний пристрій з браузером, що підтримує JavaScript, а також підключення до мережі Інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програма не потребує інсталяції і доступна на сайті за посиланням https://dirversity-demo.herokuapp.com/.

ВХІДНІ ДАНІ

Вхідна інформація для додатку:

а) навчальні матеріали;

б) метадані до навчальних матеріалів.

ВИХІДНІ ДАНІ

Вихідна інформація додатку:

а) посилання на завантажений у хмарне сховище навчальний матеріал;

б) ідентифікатор завантаженого навчального матеріалу.

ДОДАТОК Г

Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

Довідки про впровадження результатів роботи

“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 14-1

Аркушів 2

Київ – 2020 року

“Затверджую”
В.о. зав. кафедри АПЕПС
КПІ ім. Ігоря Сікорського

к.т.н. Коваль О.В.

“1” червня 2020 р.

АКТ ВПРОВАДЖЕННЯ

результатів дипломної роботи освітньо-кваліфікаційного рівня “бакалавр”
Гученко Микити Сергійовича.

Гученко М.С. в процесі виконання дипломної роботи на тему “*Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури*” розробив WEB-систему, яка дозволяє автоматизувати централізоване управління розповсюдженням та опрацюванням навчальної літератури за допомогою web-системи.

Крім того, зазначений програмний продукт є складовою комплексного програмного забезпечення з автоматизації засобів підтримки навчального процесу, що розробляється на кафедрі АПЕПС ТЕФ у межах робочого часу .

Даний програмний продукт приймається до дослідної експлуатації в науково навчальній лабораторії з моделювання динамічних процесів та систем

Науковий керівник лабораторії
к.т.н., доцент Гагарин О.О.
Завідуючій лабораторією Гайдаржи В.І.

ДОДАТОК Д

Web-система з централізованого управління розповсюдженням та опрацюванням навчальної літератури

Апробації

“КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР62134 20Б 15-1

Аркушів 3

Київ – 2020 року

УДК 621.43.056:632.15

Студент 4 курсу, гр. ТР-62 Гученко М.С.
Ст.викл. Гайдаржи В.І.

WEB-СИСТЕМА З ЦЕНТРАЛІЗОВАНОГО УПРАВЛІННЯ РОЗПОВСЮДЖЕННЯМ ТА ОПРАЦЮВАННЯМ НАВЧАЛЬНОЇ ЛІТЕРАТУРИ

Об’єми інформації та швидкість з якою, ці дані необхідно оновлювати для підтримки актуальності невпинно зростає. Технологічні проблеми навчального процесу вимагають наявності інформаційної системи для зберігання та розповсюдження навчальної літератури, що має сприяти покращенню навчального процесу. Наведемо короткий огляд згаданих технологічних проблем:

- відсутність автоматизованих засобів для викладача формування масиву навчальної літератури (наповнення електронних листів ідентифікаторами, посилань на тему з навчального плану),
- відсутність можливості заздалегідь розпланувати дану поширення матеріалів,
- відсутність контролю процесу опанування навчальної літератури,
- визначення спрямованості навчальних матеріалів отриманих студентом з плином часу,
- відсутність організаційних принципів збереження навчальних матеріалів.

Існуючі інструменти у своїй більшості пропонують рішення [1], що вимагають багато часу та ресурсів на впровадження. В процесі вивчення проблеми зроблено висновок про доцільність використання мікросервісної архітектури [2].

Звертаючи увагу на те, що заклад вищої освіти, може використовувати нестандартні рішення для зберігання даних, розроблено систему управління розповсюдженням та опрацюванням навчальної літератури на основі мікросервісної архітектури, схема якої представлена на рисунку 1. Система підвищує якість та швидкість опанування студентами навчальних матеріалів, у порівнянні з традиційними підходами.

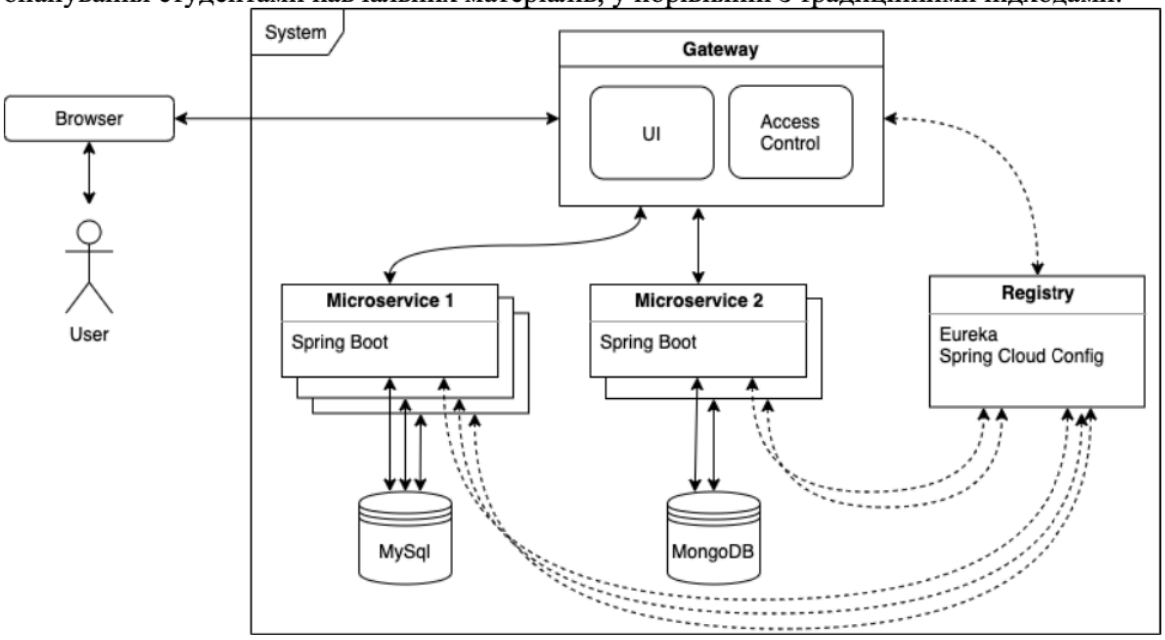

Рисунок 1. Мікросервісна архітектура розробленої системи

Перелік посилань:

1. А. М. Береза Основи створення інформаційних систем: Навч. посібник. 2 видання, перероблене і доповнене – К.: КНЕУ, 2001. – 204 с.
2. J. Carnell Spring Microservices in Action / John Carnell., 2017. – 384 с. – (In Action).

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут
імені Ігоря Сікорського»

ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ
Кафедра автоматизації проектування енергетичних процесів і систем

Д О В І Д К А

Гученко Микита Сергійович у співавторстві з Гайдаржи Володимиром Івановичем для участі у VI Міжнародній науково-практичній конференції «Сталий розвиток – XXI століття (наукові читання імені Ігоря Нєдіна)» подали статтю «Архітектура системи управління розповсюдженням та опрацюванням навчальної літератури» (обсяг 8 сторінок).

Стаття прийнята до друку і буде опублікована в колективній монографії «Сталий розвиток — XXI століття: управління, технології, моделі. Дискусії 2020».

Член організаційного комітету, відповідальна
за підготовку матеріалів Конференції Л.І. Кублій

Контактні особи:
Сегеда Ірина Василівна +38 (050) 695 54 43
Кублій Лариса Іванівна +38 (097)558 27 17
E-Mail: nedin.conf@gmail.com